My Understanding of "Latch Clustering for Timing-Power Co-Optimization"

ACM/IEEE Design Automation Conference (DAC2020)

Chau-Chin Huang, Gustavo Tellez, Gi-Joon Nam and Yao-Wen Chang

2021-06-02
Presenter: Sethupathi Balakrishnan

The Electronic Design Automation Laboratory
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei 106, Taiwan

臺灣大學

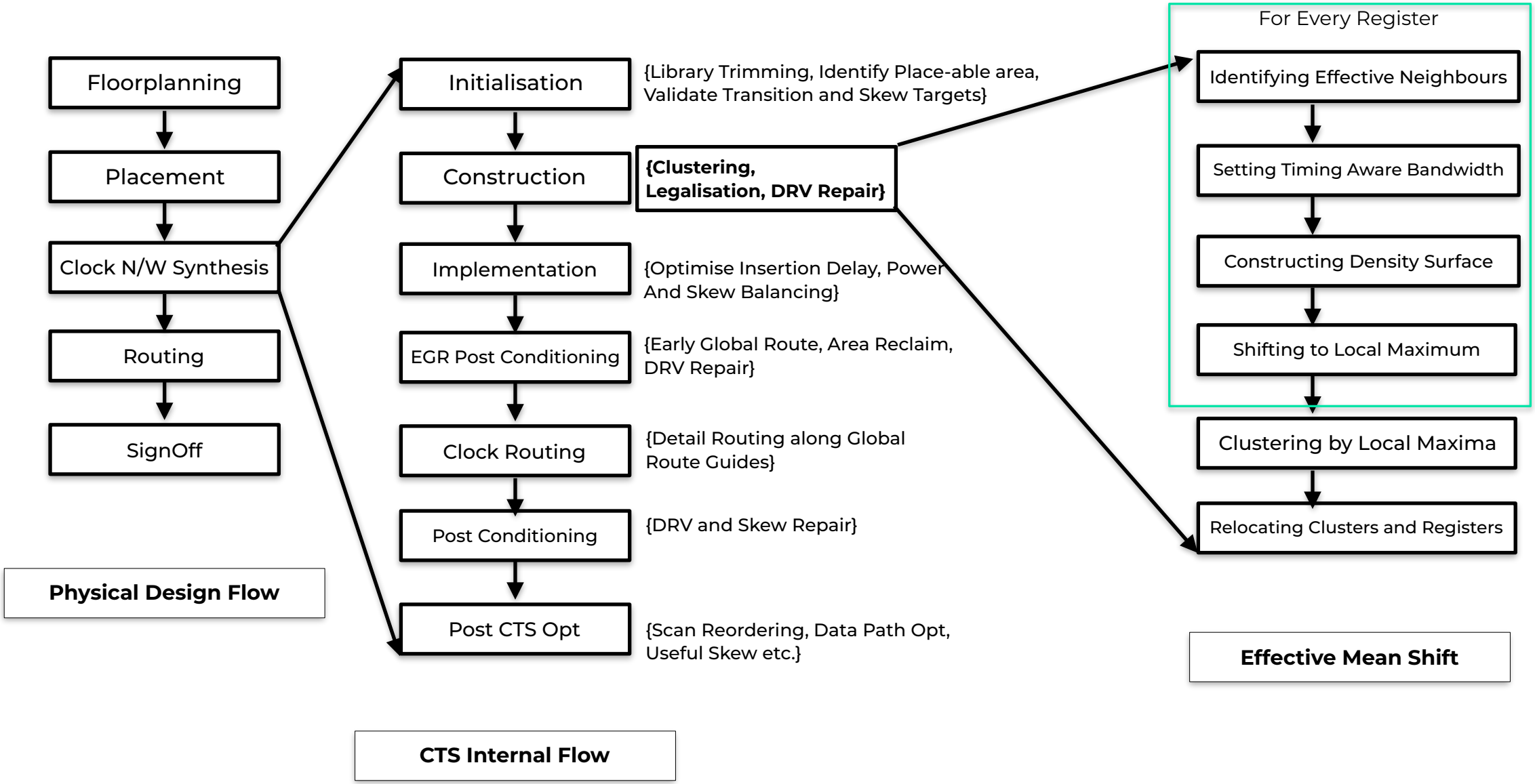
Presentation Outline

- **1. Why is this work relevant/important?**
- **2. Previous Work to this Work (Register Clustering)**
- **3. This Work (Latch Clustering)**
- **4. Results/Conclusions**
- **5. Appendix 1 — Physical Design Flow (Designer PoV)**
- **6. Appendix 2 — Static Timing Analysis**

Why is this work relevant/important?

- "**Clock Sink Point Clustering** is still a problem not solved well enough". — Prof. AB Kahng
- This problem is still being investigated by people from Qualcomm San Diego, Cadence Innovus R&D Group - San Jose, Synopsys CTS Group, Prof. Jiang, Prof. Yao-Wen.
- Existing tools still do not give optimal PPA i.e The designer has to do more work in CTS design step. For eg. Clock Sink clustering in Multi-Source CTS in Physical Design Flow.
- A well clustered CTS can make a lot difference in NFE's, TNS and WNS.
- The design converges well with optimal clustering.
- Reference — Open cases from Synopsys Solvnet —

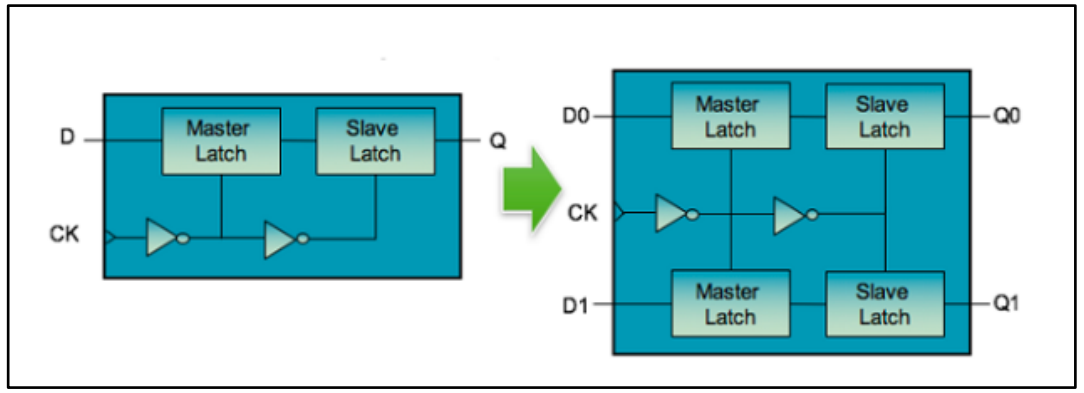
Register Clustering — Previous Work



Register Clustering (Previous Work Contd.)

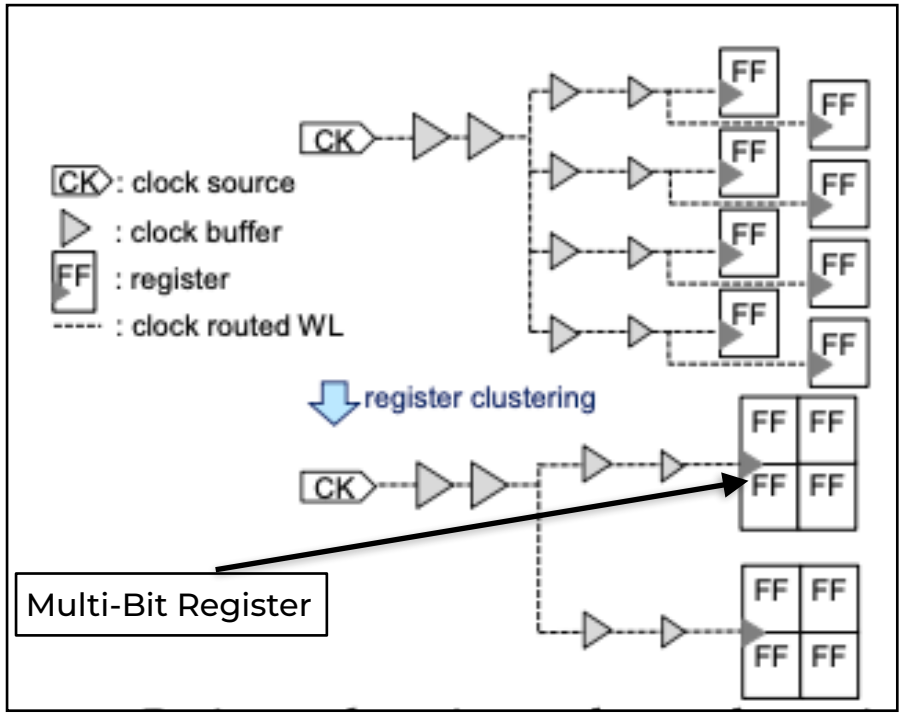
- Register Clustering can save power and improve timing by
- 1. Reducing Clock Route Length
- 2. Reducing no. of buffers used for CTS.
- 3. Reducing the no. of clock sink points and
- therefore reducing the pin capacitance.

- **MultiBit Registers** — Formed by merging single-bit flops.
- Internal Clock inverters are shared.



Conventional Techniques —

1. Voltage Islands
2. Clock Gating
3. Optimal Gate Mapping etc.
4. Power Opt. In Architecture Level etc.



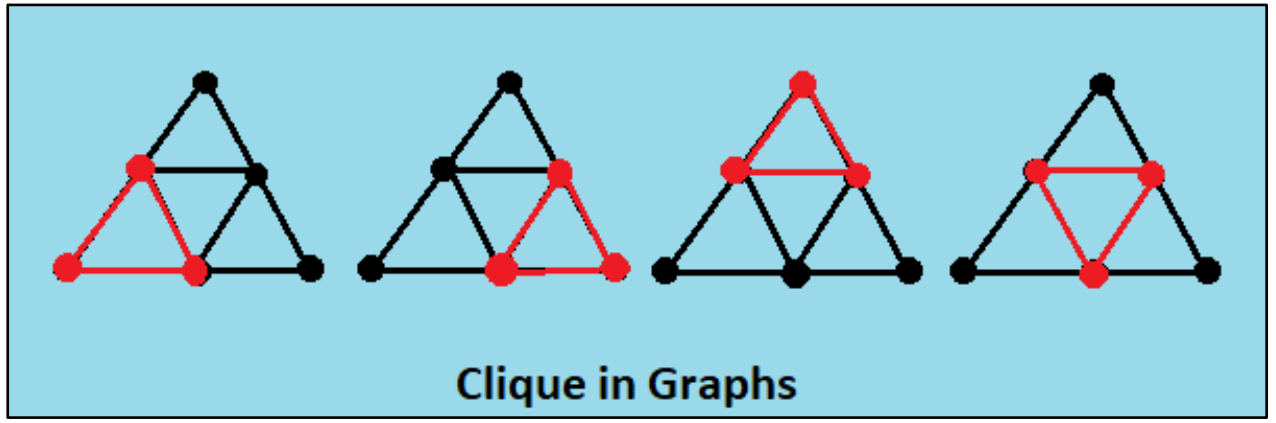
Register Clustering (Previous Work Contd.)

- Two Types —
- **1. Clique Partitioning**
- **2. K-means Clustering**

In graph theory, a **comparability graph** is an undirected graph that connects pairs of elements that are comparable to each other in a partial order.

- **Clique Partitioning** — Constructs a **compatibility graph** (recording the clustering compatibility b/w any two registers based on their timing feasible regions) and then extracts **maximal cliques** to form Multi-Bit registers without timing degradation.

A **clique** is a subset of vertices of an undirected graph G such that every two distinct vertices in the clique are adjacent; that is, its induced subgraph is complete.
A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex, that is, a clique which does not exist exclusively within the vertex set of a larger clique.



K-Means Clustering

- Relaxes Timing constraints to maximum displacement constraints, trying to minimise impact on timing. K-means, however, is sensitive to initializations and outliers (distant from other registers); it starts with a prespecified number of clusters and initial cluster centers (seeds), iteratively assigns registers to nearest clusters, and finally converges to a local minimum of within-cluster total displacement.

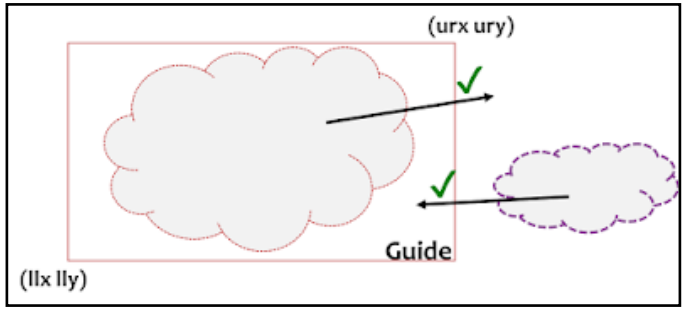
-
```
create_clock -period 20 -waveform {0 6} -name SYS_CLK  [get_ports SYS_CLK]
create_generated_clock -divide_by 2 -source [get_ports sys_clk] -name gen_sys_clk [get_pins UFF/Q]
set_clock_latency 1.86 [get_clocks clk250]
set_clock_uncertainty -setup -rise -fall 0.2 [get_clocks CLK2]
set_clock_transition -min 0.5 [get_clocks SERDES_CLK]
set_clock_transition -max 1.5 [get_clocks SERDES_CLK]
set_input_delay -clock virtual_mclk 2.5 [all_inputs]
set_output_delay 0.40 [all_outputs]
set_max_delay -from [get_clocks FIFOCLK] -to [get_clocks MAINCLK] 3.5
```

Clock constraints example

Placement constraints example

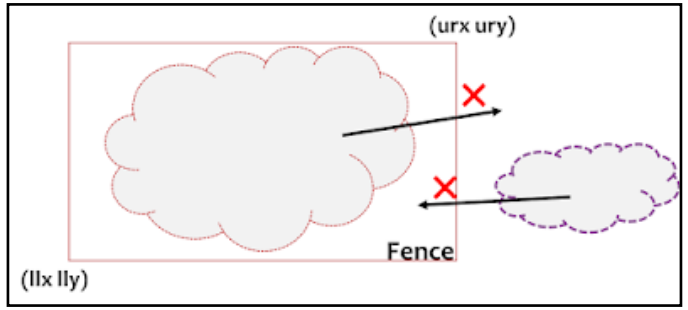
Guide, Fence and Region

Max displacement, cell padding (for congestion), target density,



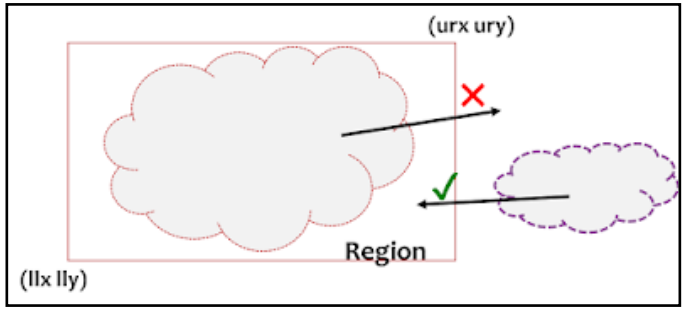
- The guide is assigned with certain cells in the design
- The guide allows to assigned cell sit outside the box
- It also allows the other cells to sit inside the box
- It is a soft constraint.

createGuide adder9 100.000 200.000 110.000 210.000



- The fence is assigned with certain cells in the design.
- A fence does not allow the assigned cell to sit outside the box defined.
- A fence does not allow the other cells to sit inside the box also. So the area is exclusively reserved for the assigned cells.
- It is a hard constraint.

createFence SH12 200 400 220 440



- The region is assigned with certain cells in the design.
- A region does not allow the assigned cell to sit outside the box defined.
- It may cause congestion in the area assigned if not chosen the area wisely.
- The only difference between the region and fence is that it allows the other cells to sit inside the box.
- It is a hard constraint

createRegion SH7 200 400 220 440

K Means Clustering — Contd.

Consider a timing-optimized placement as the input. Creating large clusters or dragging outliers far away inevitably causes large disruption to placement thus incurring significant timing degradation. The more timing degradations, the more timing ECO efforts. Besides, once registers are clustered (even few), we can save clock power.

Based on these investigations, a good register clustering algorithm is desired

1) to require no prespecified number of clusters,

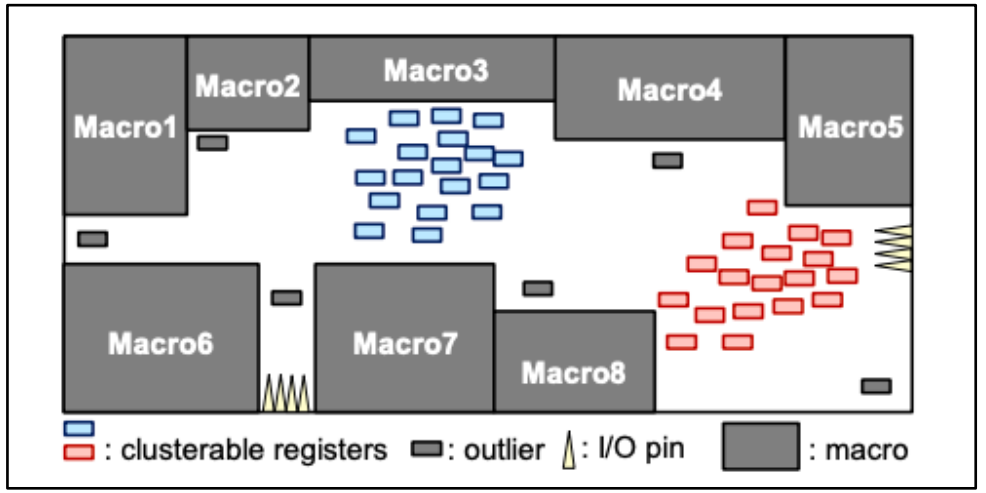
2) to be insensitive to initializations,

3) to be robust to outliers,

4) to be tolerant of various register distributions,

5) to be efficient and scalable, and

6) to balance power and timing.



. These problems are addressed in the paper —

. **"Graceful Register Clustering by Effective Mean Shift Algorithm for Power and Timing Balancing*"** —

. Effective mean shift **augments classic mean shift** with **special treatments for register clustering** to attain these goals.

.

K Means Clustering (Unsupervised Learning)

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the K Means Algorithm.
The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.The algorithm works as follows:

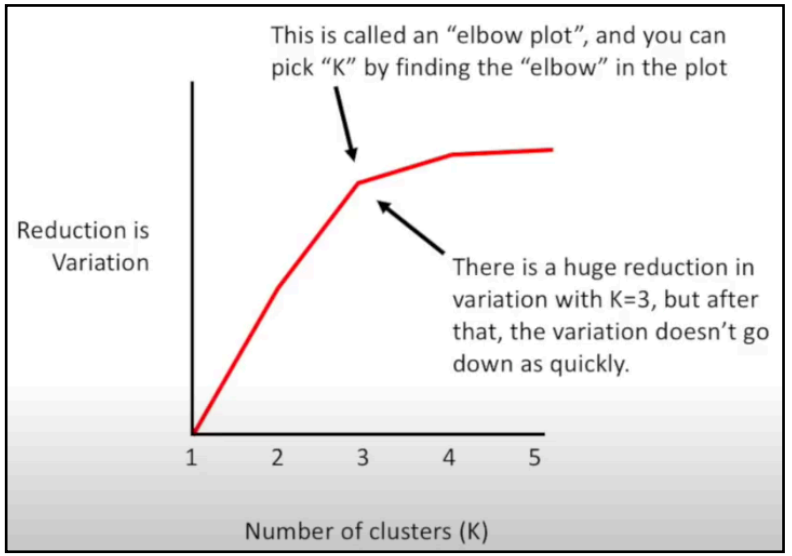
1. First we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

Best "K" is arrived by iterating over various 'k' values and finding out the variation in each iteration.

Pseudocode

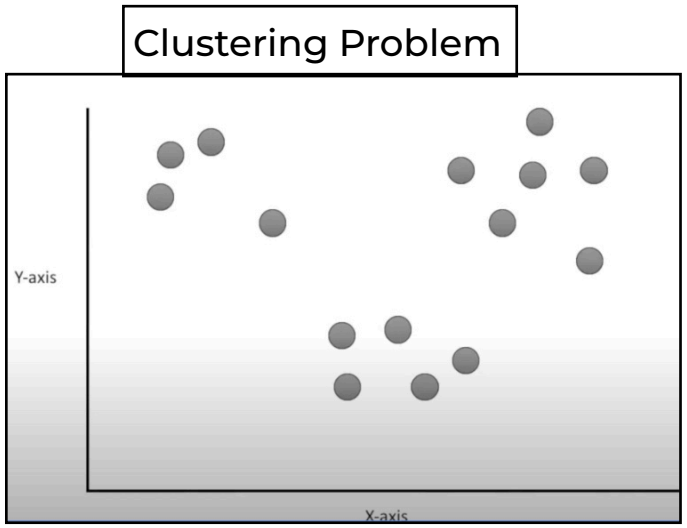
```
Initialize k means with random values

For a given number of iterations:
    Iterate through items:
        Find the mean closest to the item
        Assign item to mean
        Update mean
```

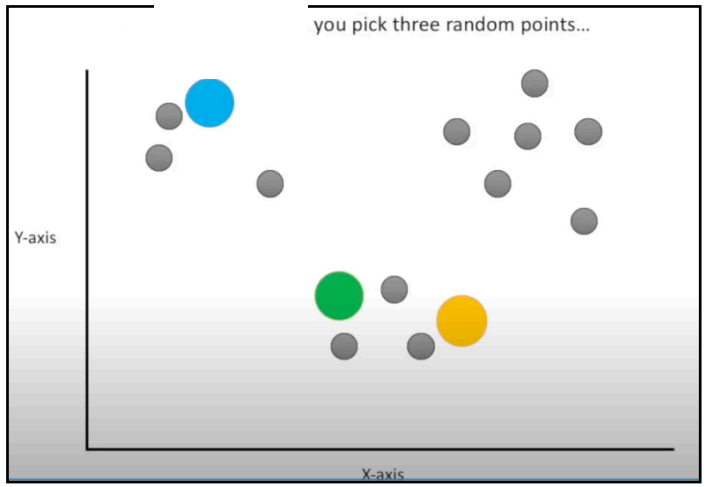
Average value of all items in the cluster

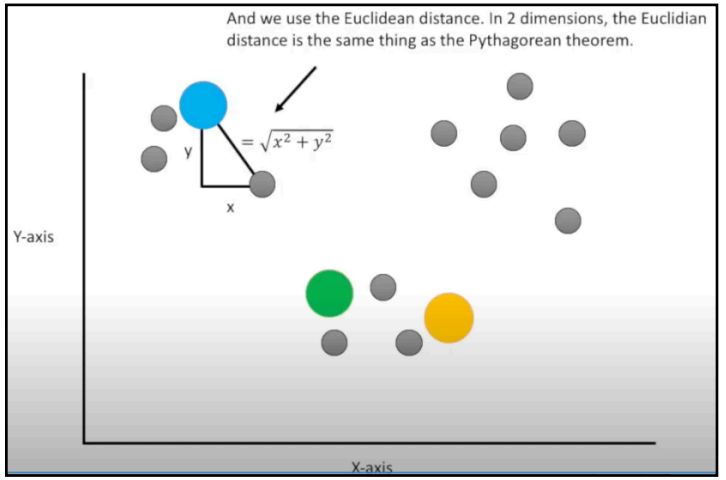


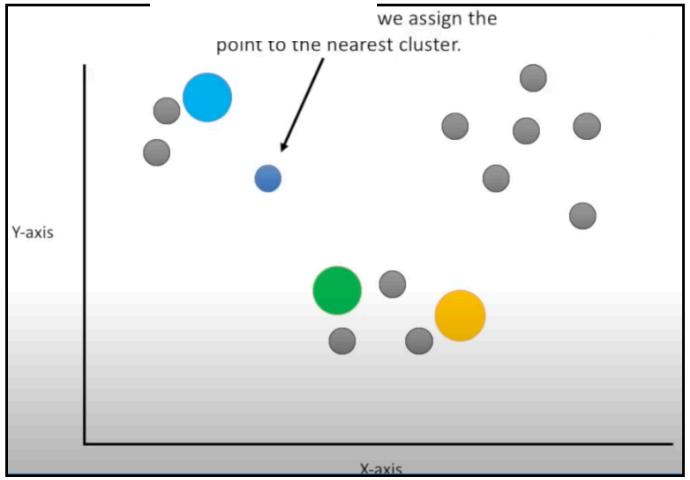
How to arrive at optimal k?

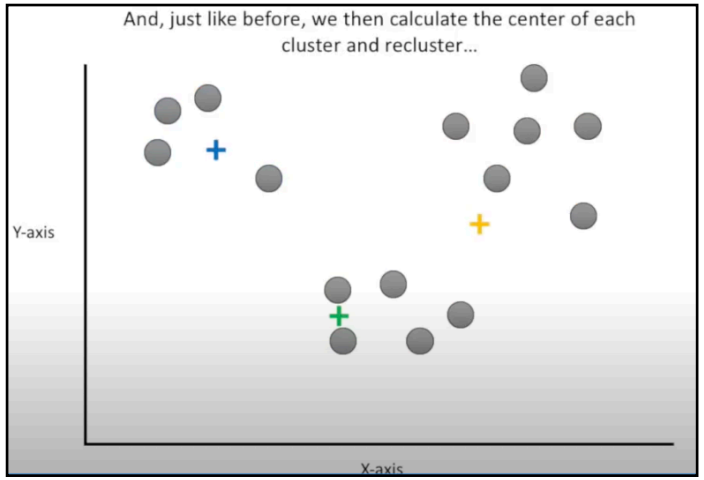
K Means Clustering

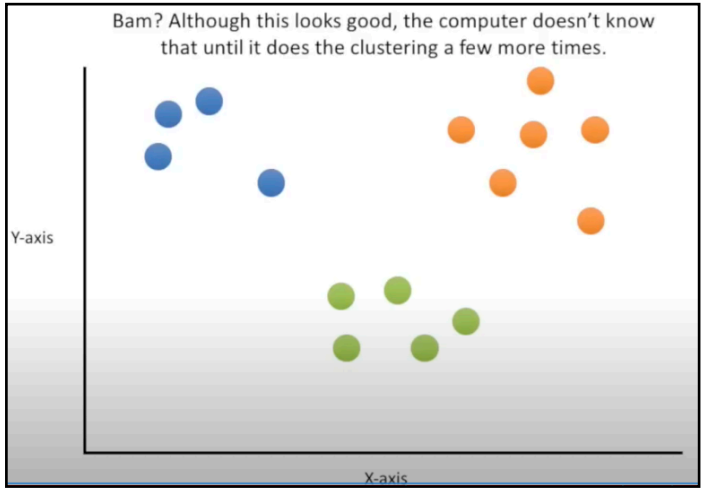












More Info —http://vision.stanford.edu/teaching/cs131_fall1617/lectures/lecture13_kmeans_mean_shift_cs131_2016

Solution after several iterations

Mean Shift Clustering

Meanshift is falling under the category of a clustering algorithm in contrast of Unsupervised learning that assigns the data points to the clusters iteratively by shifting points towards the mode (mode is the highest density of data points in the region, in the context of the Meanshift). As such, it is also known as the **Mode-seeking algorithm.**

Given a set of data points, **the algorithm iteratively assigns each data point towards the closest cluster centroid and direction to the closest cluster centroid is determined by where most of the points nearby are at.** So each iteration each data point will move closer to where the most points are at, which is or will lead to the cluster center. When the algorithm stops, each point is assigned to a cluster.

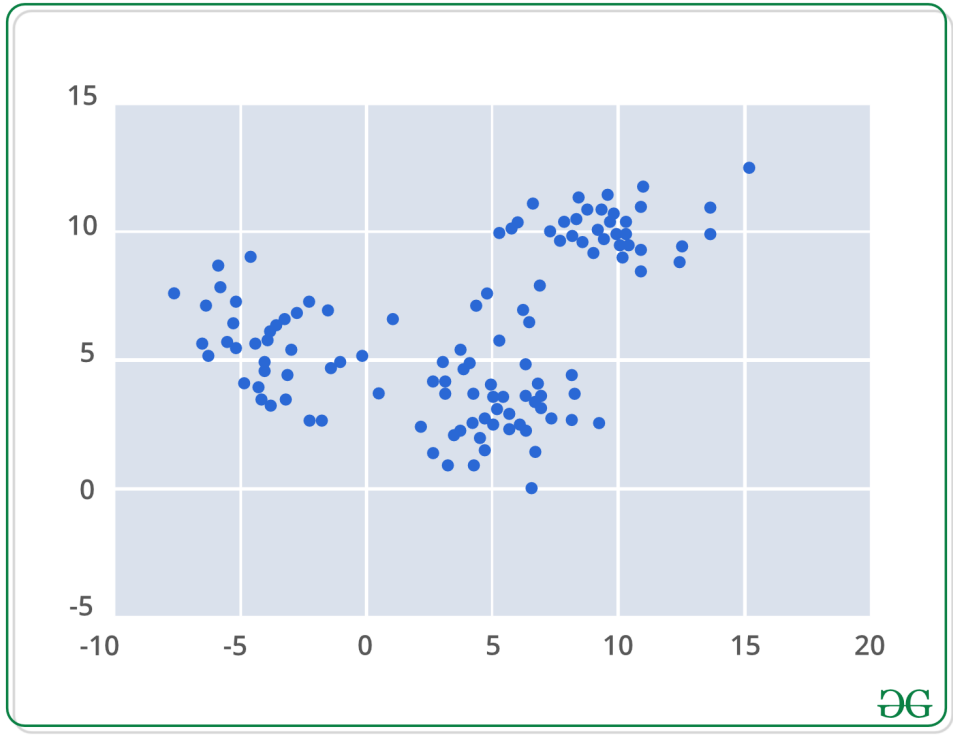
Unlike the popular K-Means cluster algorithm, mean-shift does not require specifying the number of clusters in advance. The number of clusters is determined by the algorithm with respect to the data. **O(n^2)**

Kernel Density Estimation

1. The first step when applying mean shift clustering algorithms is representing your data in a mathematical manner.

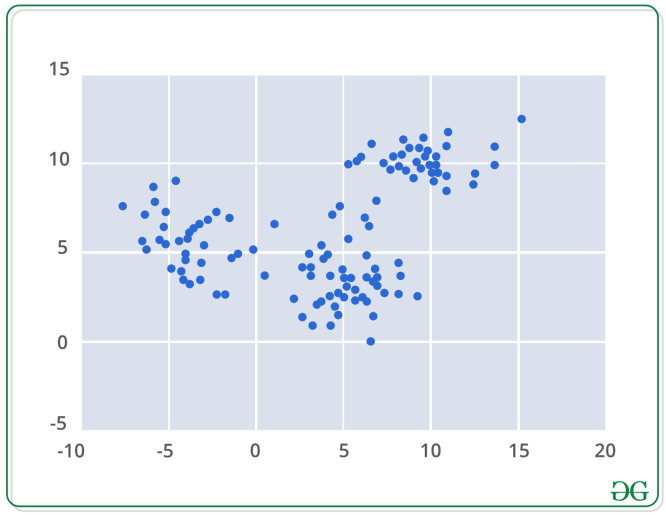
2. Mean-shift builds upon the concept of **kernel density estimation** is sort KDE. Imagine that the data was sampled from a **probability distribution (Guassian Curve)**. KDE is a method to estimate the underlying distribution also called the probability density function for a set of data.

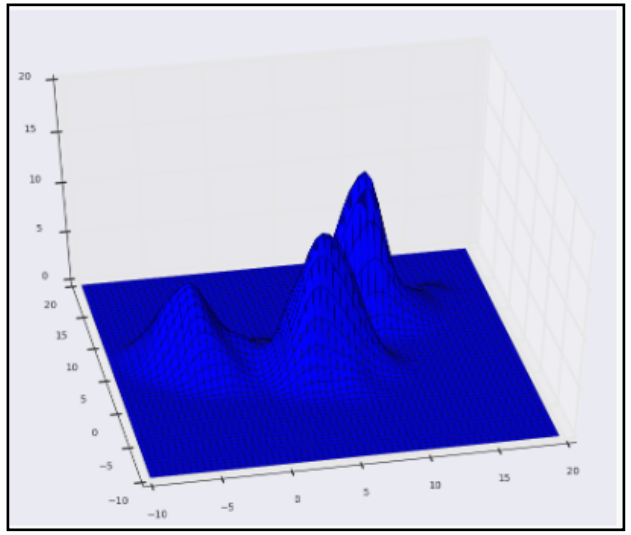
3. It works by placing a kernel (weighting function) on each point in the data set. Adding up all of the individual kernels generates a probability surface example density function. Depending on the kernel bandwidth parameter used, the resultant density function will vary..

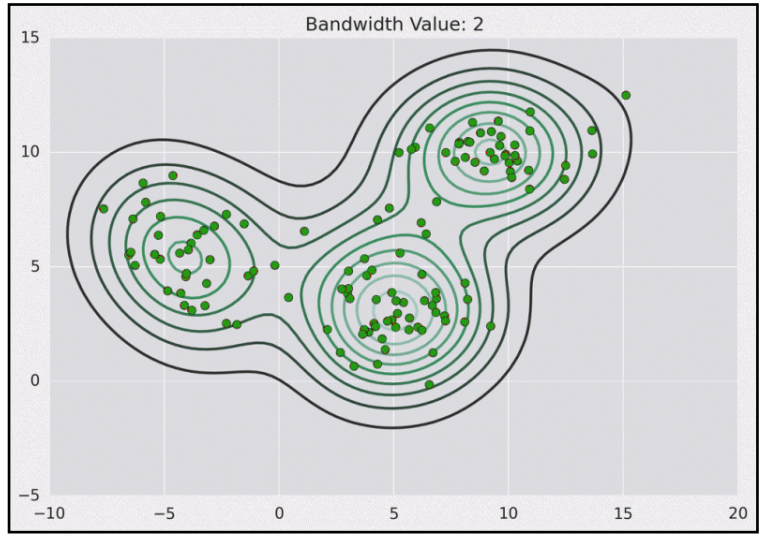


Kernel Density Estimation — Contd..

It works by placing a kernel (weighting function) on each point in the data set. **Adding up all of the individual kernels generates a probability surface example density function.** Depending on the **kernel bandwidth** parameter used, the resultant density function will vary.







$$f_K(\mathbf{u}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{u} - \mathbf{u}_i}{h}\right)$$

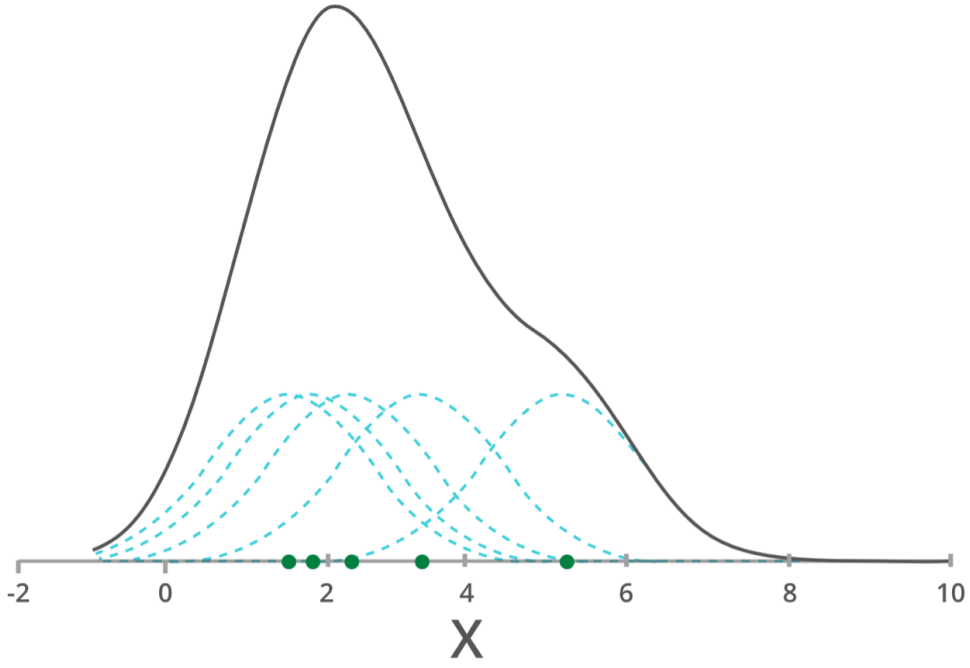
K — Kernel
h — Bandwidth Parameter (Radius of the kernel)
fK(u) — Kernel Density

Kernel Density Estimation — Contd..

$$f_K(\mathbf{u}) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{\mathbf{u} - \mathbf{u}_i}{h})$$

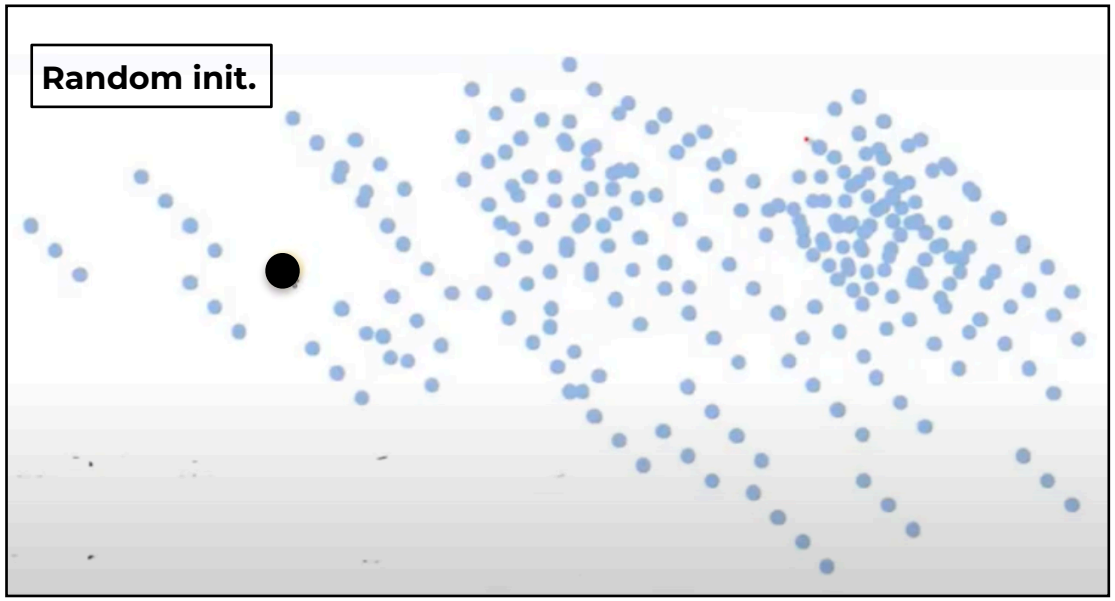
K — Kernel
h — Bandwidth Parameter
fK(u) — Kernel Density Estimation has to satisfy the following two conditions —

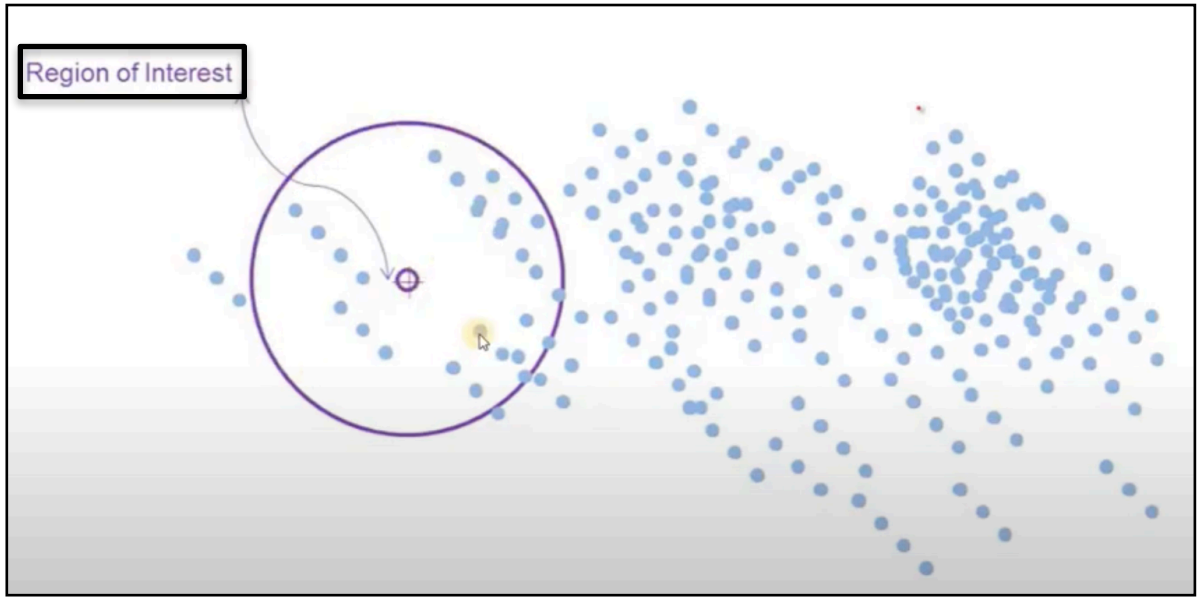
Below we plot an example in one dimension using the Gaussian kernel to estimate the density of some population along the x-axis. We can see that each sample point adds a small Gaussian to our estimate, centered about it and equations above may look a bit intimidating, but the graphic here should clarify.

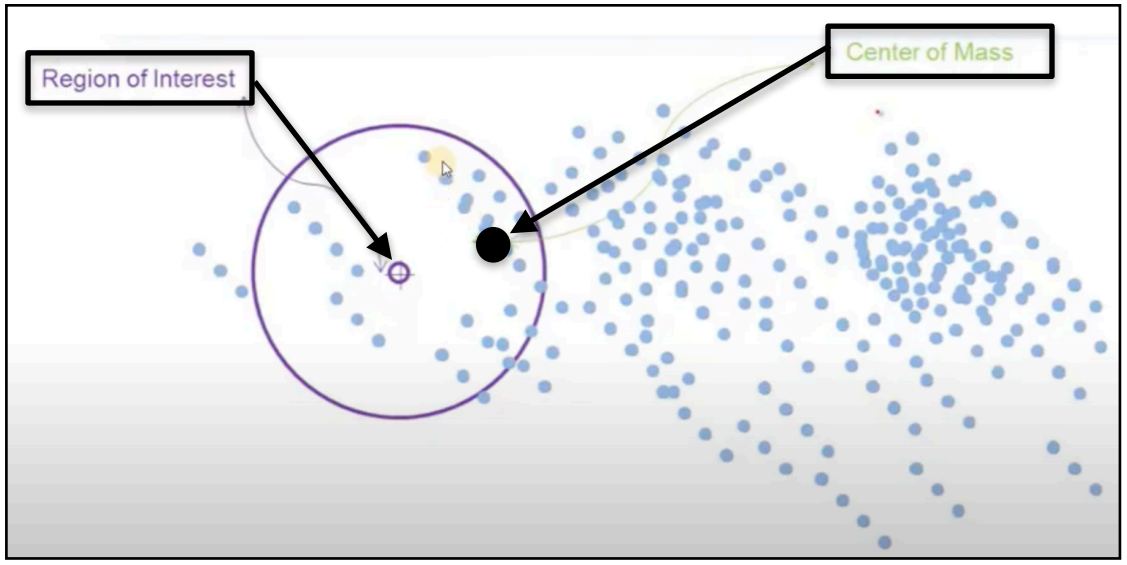


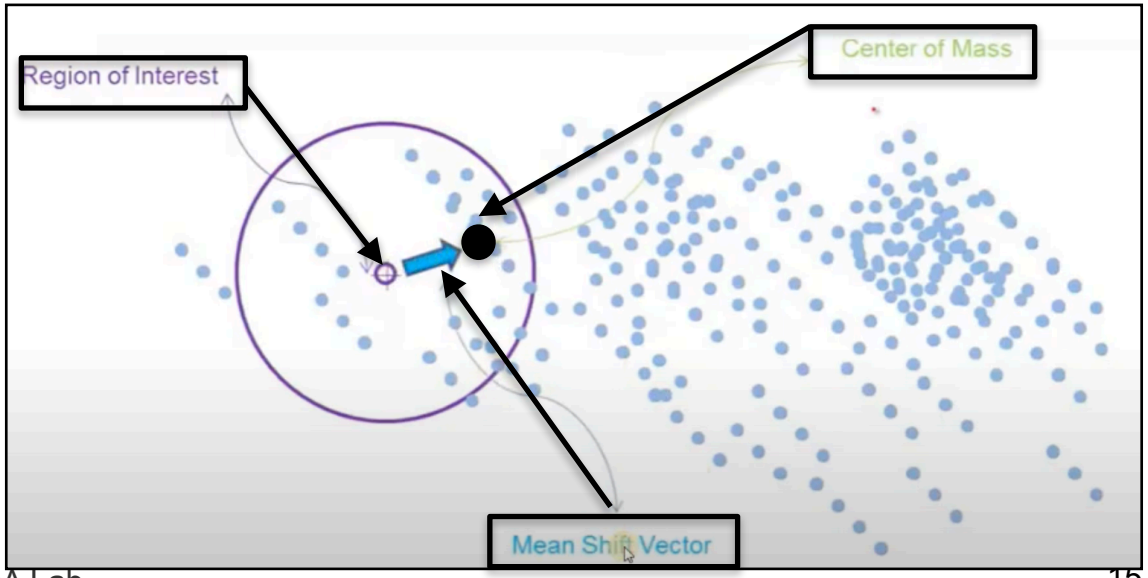
Example of kernel density estimation using a gussain kernel for each data point: Adding up small Gauggians about each example returns our net estimate for the total density, the black curve.

Visualisation/Intuition for better clarity

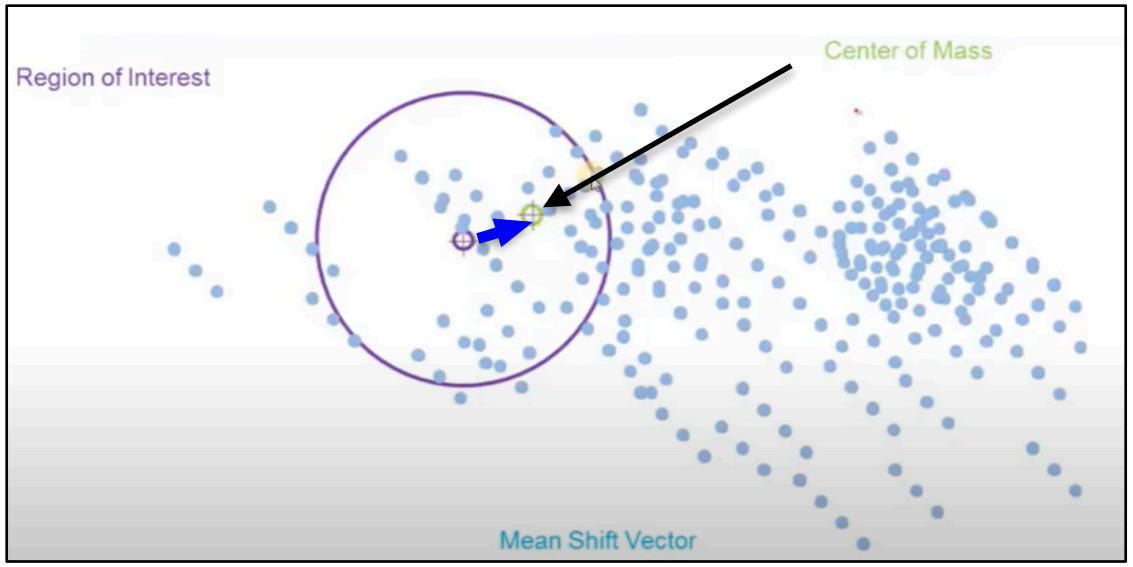


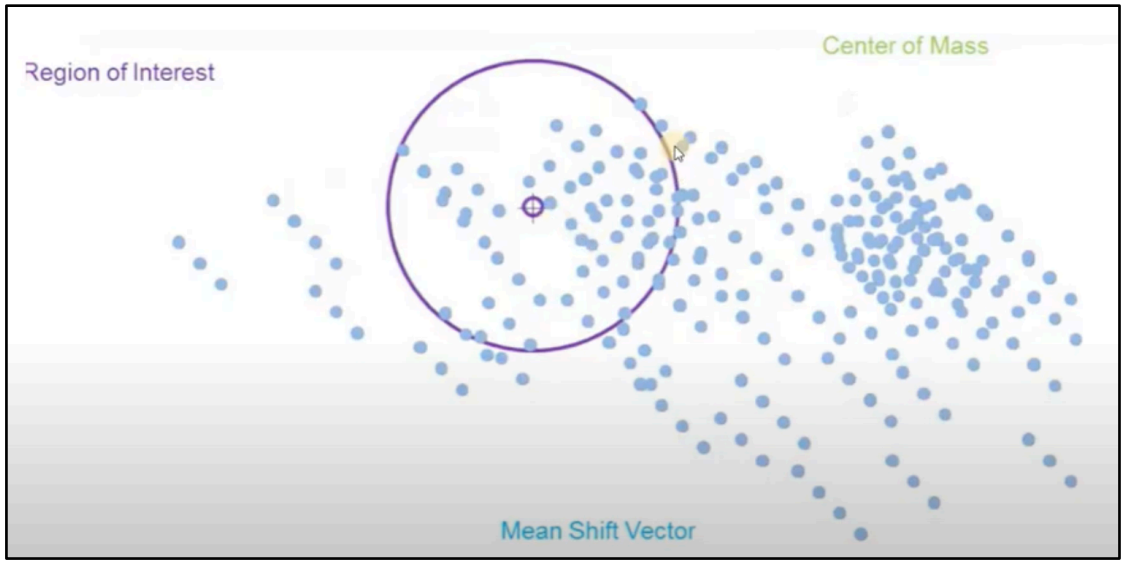


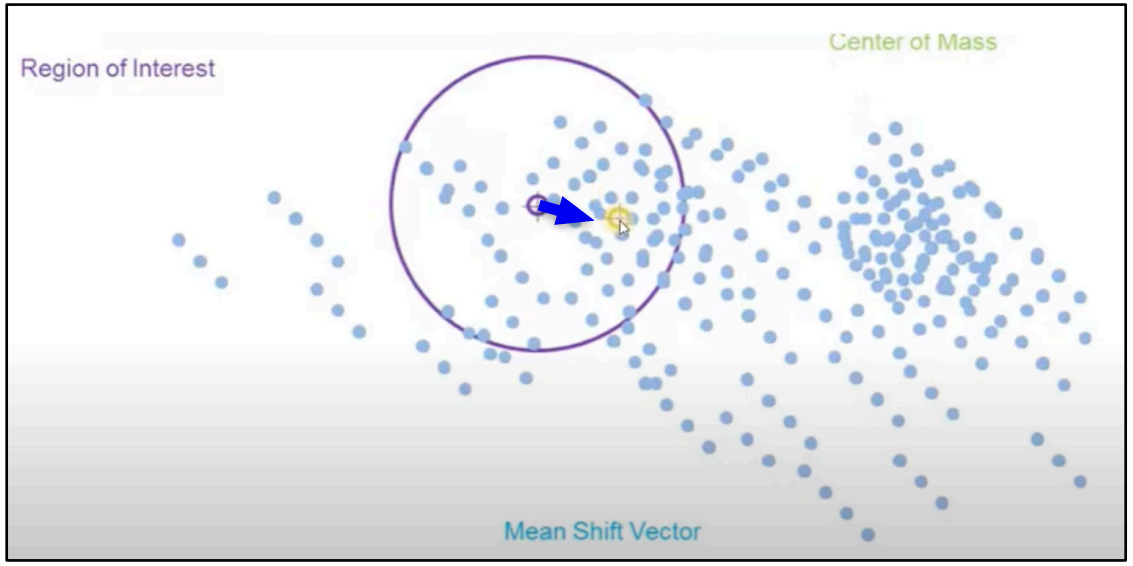


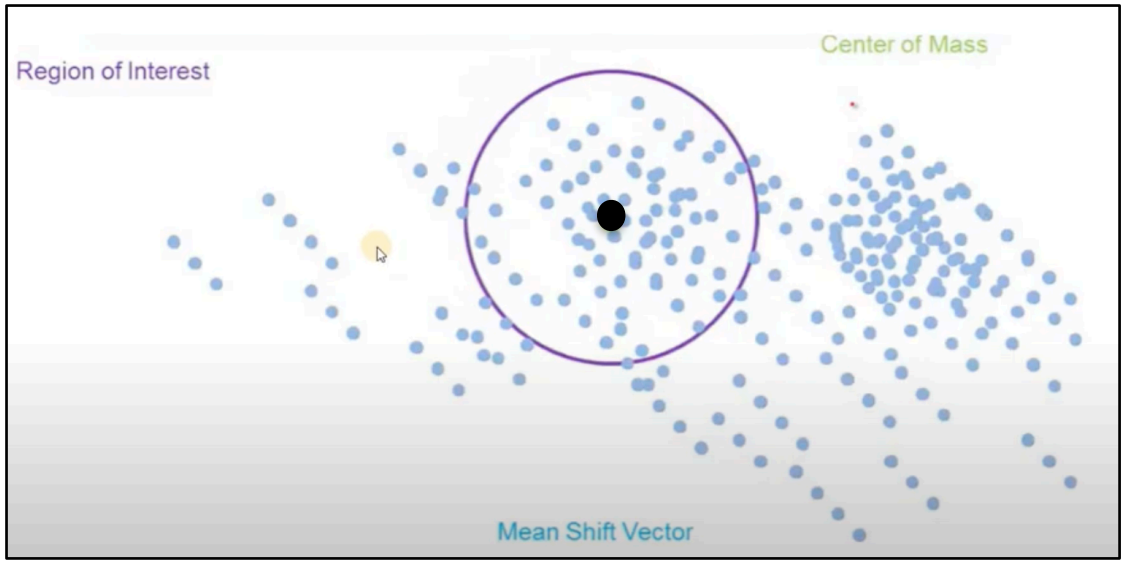


Visualisation contd..

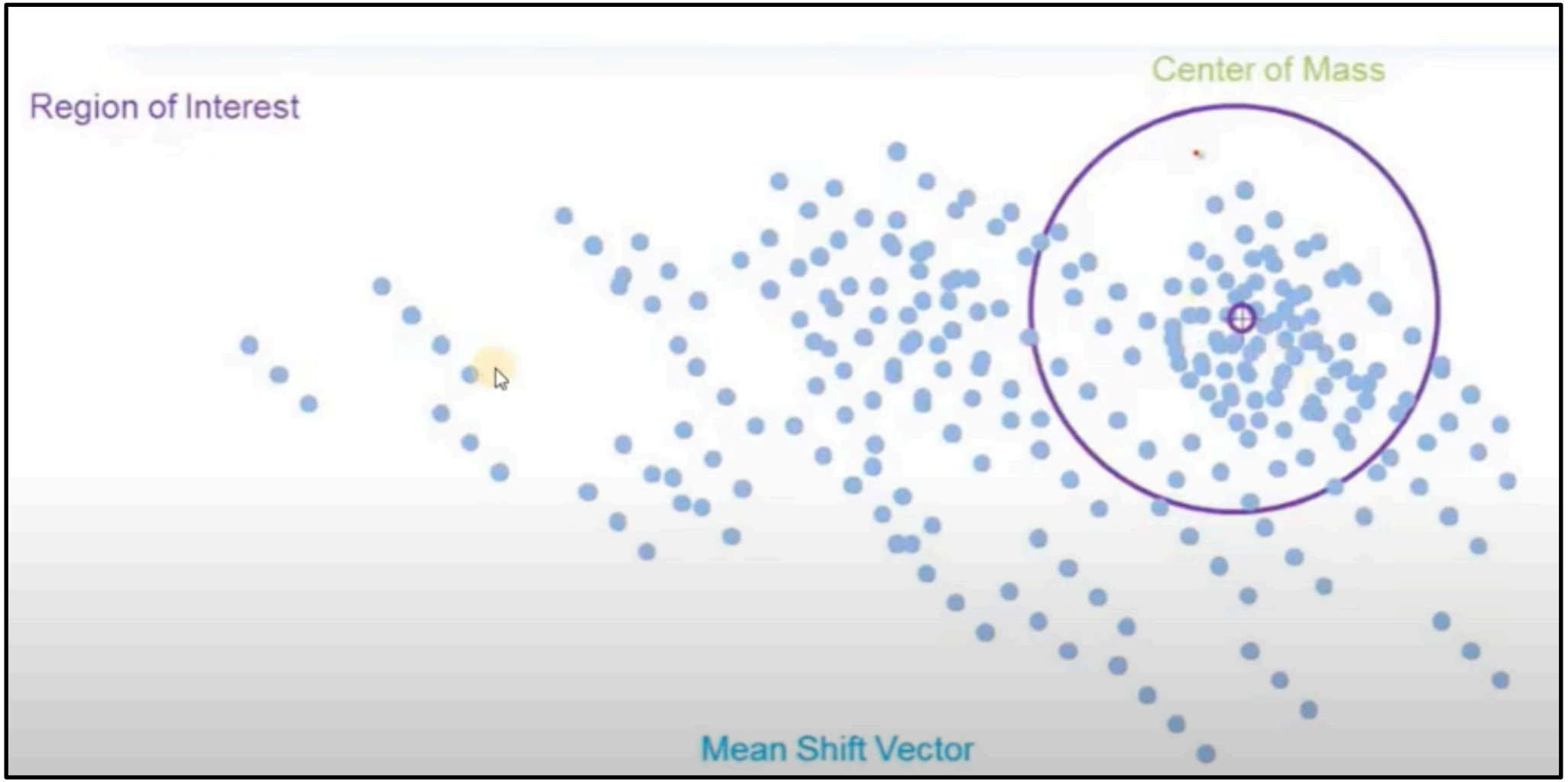




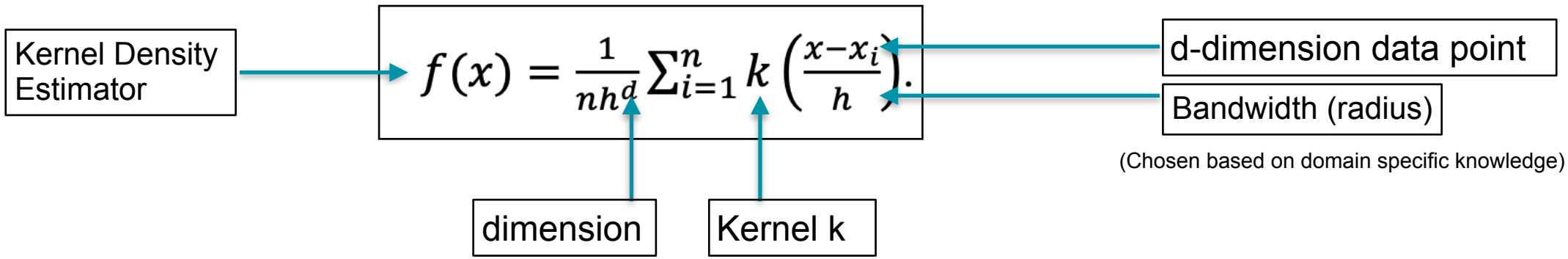




Final Result



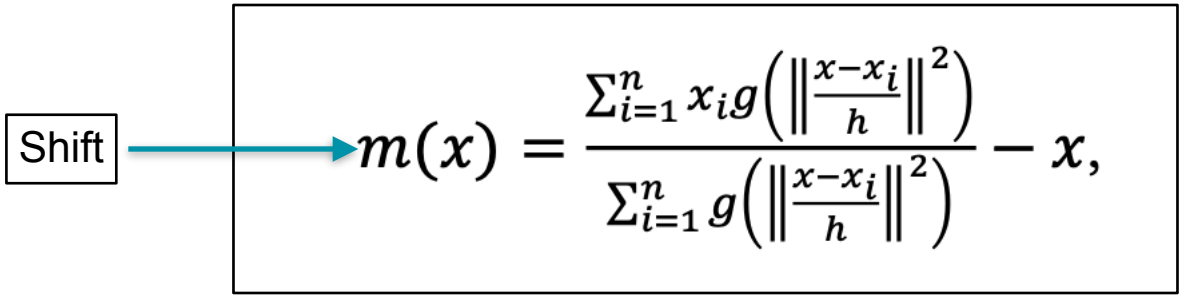
Technicalities



—The algorithm starts with making a copy of the original data points and freezing the original ones.

—The copied points are iteratively shifted against the original frozen points.

—The shift m of each point is computed by performing gradient ascent on the density function until it converges to a stationary point.



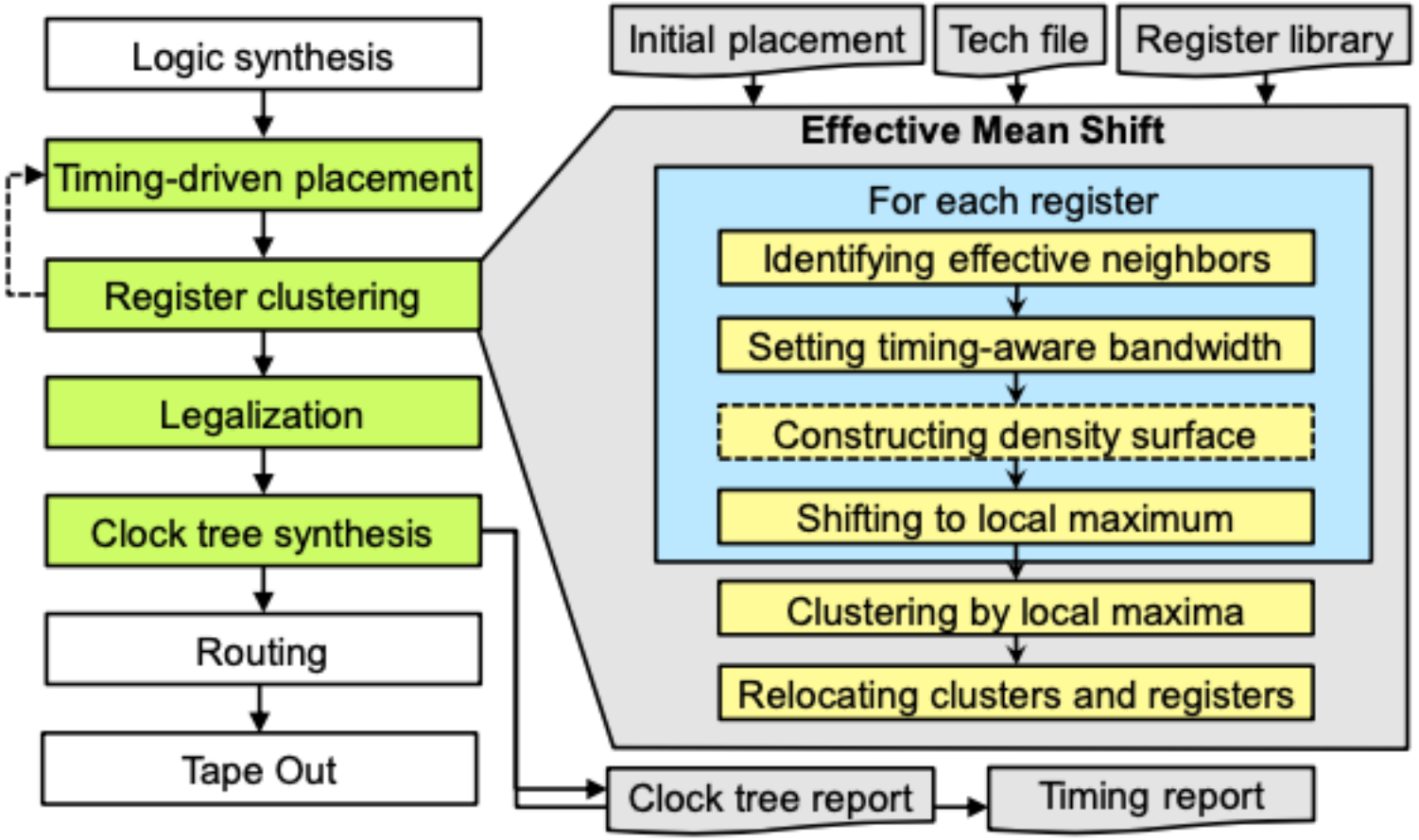
$k(x) = \kappa(\|x\|^2)$, $\|x\|^2$ means squared Euclidean distance, and gradient $g(x) = -\kappa'(x)$. $m(x)$ points towards the direction of maximum increase in density.

Finally, all points associated with the same stationery point belong to the same cluster.

Problem — Complexity O(Tn^2);

T = No. Of Iterations, n = no. of data points

Effective Mean Shift Clustering



Effective Mean Shift Clustering

- PROBLEM FORMULATION AND METHODOLOGY —
- Inputs to Problem — 1. Timing Optimized Placement
  2. Multi-Bit Register Library
  3. User Defined Maximum Allowable Displacement

- Goal — Save clock power without placement/timing disruption by minimizing the total sum of register displacement and no. of clusters, while satisfying the cluster size constraint and maximum displacement constraints.

Technicalities

Classic Mean Shift	Adaptive Mean Shift (Variable Bandwidth)	Effective Mean Shift
Density estimator $\frac{1}{nh^d}\sum_{i=1}^{n} k\left(\frac{x-x_i}{h_i}\right)$	$\frac{1}{n}\sum_{i=1}^{n} \frac{1}{{h_i}^d} k\left(\frac{x-x_i}{h_i}\right)$	$\frac{1}{n}\sum_{i\in KNN'(x)} \frac{1}{{h_i}^d} k\left(\frac{x-x_i}{h_i}\right)$
Shift point $\frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}$	$\frac{\sum_{i=1}^{n} \frac{x_i}{{h_i}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}{\sum_{i=1}^{n} \frac{1}{{h_i}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}$	$\frac{\sum_{i\in KNN'(x)} \frac{x_i}{{h_i}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}{\sum_{i\in KNN'(x)} \frac{1}{{h_i}^{d+2}} g\left(\left\|\frac{x-x_i}{h_i}\right\|^2\right)}$

1. $k(x) = \kappa(\|x\|^2)$, Gaussian kernel 2. $g(x) = -\kappa'(x)$ 3. $d = 2$

Classic Mean Shift Revisited

- Effective mean shift naturally forms clusters according to register distributio
- First, the register distribution is mapped to a density surface; dense regions form hills.
- Each register climbs up (shifts) to the nearest peak in a specified search window

- The search window (bandwidth) of each register varies and reflects its timing criticality and local density/sparsity.
- Furthermore, for efficiency and stability, we propose to consider effective neighbors via k-nearest neighbors (KNN) during iterative shift vector computation.
- Subsequently, we reassign registers and relocate clusters to further improve displacement (for timing) and refine cluster count (for power).

Classic Mean Shift Revisited & Effective Mean Shift

- CLASSIC MEAN SHIFT — (Same Bandwidth for all registers)
- Effective mean shift naturally forms clusters according to register distribution.
- First, the register distribution is mapped to a density surface; dense regions form hills.
- Each register climbs up (shifts) to the nearest peak in a specified search window

- EFFECTIVE MEAN SHIFT — (change in bandwidth for each register)
- The search window (bandwidth) of each register varies and reflects its timing criticality and local density/sparsity.
- Furthermore, for efficiency and stability, we propose to consider effective neighbors via k-nearest neighbors (KNN) during iterative shift vector computation.
- Subsequently, we reassign registers and relocate clusters to further improve displacement (for timing) and refine cluster count (for power).

Effective Mean Shift Contd.. (Bandwidth Allocation)

. 2 cases —

. 1. Tall and skinny kernel for each register (Small Bandwidth)(Density surface has a peak at every point)

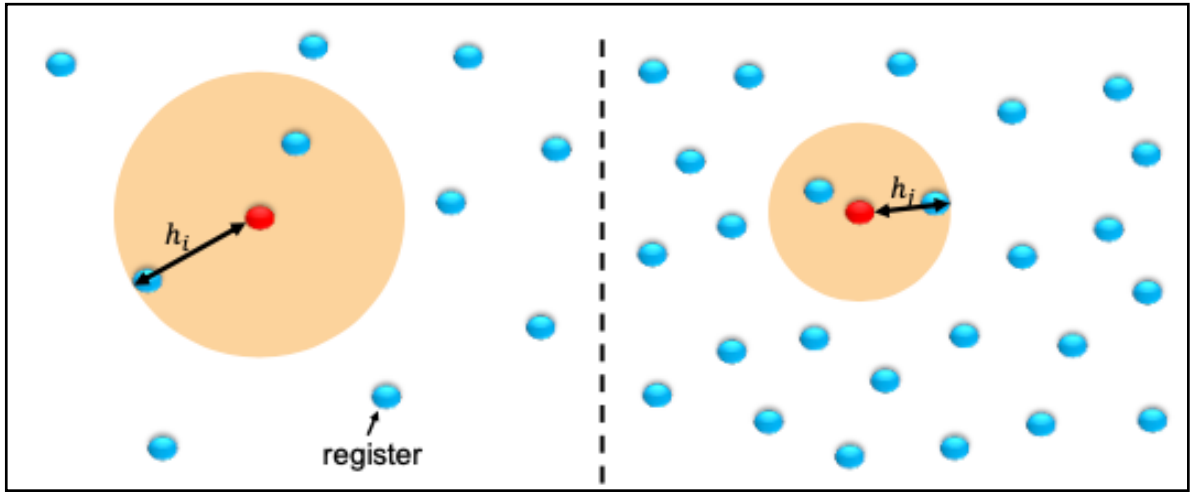
. 2. Short and fat kernel for each register (Large Bandwidth) (Smooth density surface with just one peak where all points climb up)

For a register lying in dense regions, we select a small bandwidth, thus identifying the local maximum quickly in a narrow neighborhood and avoiding a large cluster size. Hence, considering local distribution, bandwidth is first set to as the distance to its M-th nearest neighbor (see Fig). Furthermore, considering the timing criticality and maximum allowable displacement, the bandwidth of register i is

.

$$h_i = \min\left(h_{\max}, \alpha \left\| x_i - x_{i,M} \right\|\right),$$

where h_{max} denotes the maximum allowable displacement, $\|x_i - x_{i,M}\|$ means the Euclidean distance between register i and its M-th nearest neighbor ($x_{i,0} = x_i$), and α is a timing criticality coefficient; $\alpha \to 0$ for the most critical register (i.e., a very tall and skinny kernel).



Identifying Effective Neighbors

- Classic mean shift considers all original data points during shift vector computation.
- However, the points that correspond to the tails of the underlying density function receive small weights and discarded.
- For achieving this goal, we identify effective neighbors via KNN, $K \ll n$. In addition, registers belonging to KNN of a register but beyond the maximum allowable displacement are also excluded (see Fig. 6). Hence, for a register at x_j, we consider the following set of registers during the computation:

$$i \in KNN(x_j) - \{x_{j,m} \mid \|x_j - x_{j,m}\| > h_{\max}, m \leq K\}$$

$$= KNN'(x_j).$$

The density function can be rewritten as:

$$f(x) = \frac{1}{n}\sum_{i \in KNN'(x)} \frac{1}{{h_i}^d} k\left(\frac{x - x_i}{h_i}\right).$$

The shift vector becomes:

$$m(x) = \frac{\sum_{i \in KNN'(x)} \frac{x_i}{{h_i}^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)}{\sum_{i \in KNN'(x)} \frac{1}{{h_i}^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)} - x.$$

Effective Mean Shift

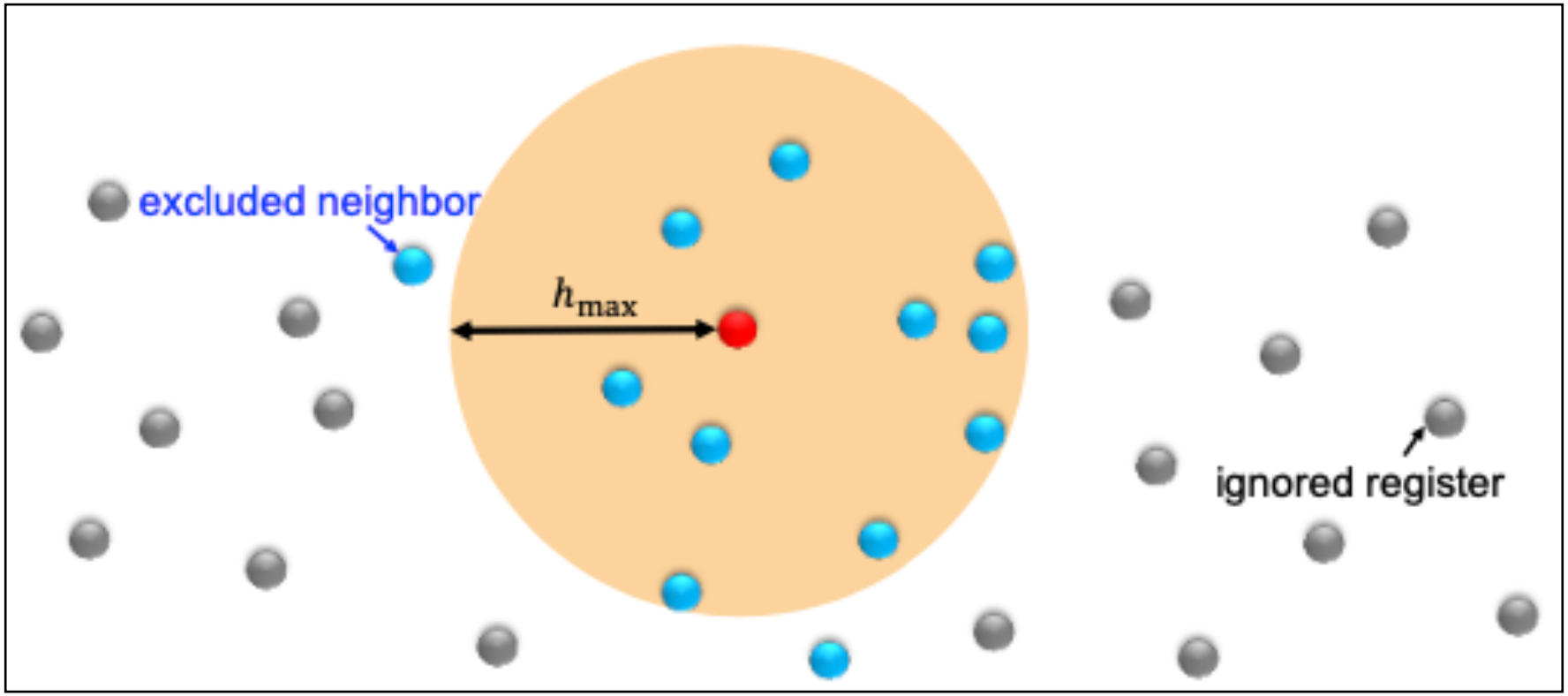
$$f(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{{h_i}^d} k\left(\frac{x - x_i}{h_i}\right).$$

The shift vector becomes:

$$m(x) = \frac{\sum_{i=1}^{n} \frac{x_i}{{h_i}^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)}{\sum_{i=1}^{n} \frac{1}{{h_i}^{d+2}} g\left(\left\|\frac{x - x_i}{h_i}\right\|^2\right)} - x.$$

Classic Mean Shift

Effective Neighbors Identified by KNN (K=12)



Shifting to Local Density Maxima

. After identifying effective neighbors and selecting a proper bandwidth for each register, we construct the density surface. We make a copy of the original register coordinates and freezing the original ones.

The copied coordinates $\{y^t\}$ (t denotes the j iteration index) are iteratively shifted against the original frozen points $\{x_j\}$. Hence, each register undergoes the following steps to seek the local density maximum.

.

1. Set the initial coordinates, $y_j^0 = x_j, j = 1..n$.
2. Identify effective neighbors, $KNN'(y_j^0)$; set bandwidth h_j.
3. Compute the mean shift vector $m(y_j^t)$ by Equation (8).
4. Shift each register,

$$y_j^{t+1} = y_j^t + m(y_j^t) = \frac{\sum_{i \in KNN'(y_j^0)} \frac{x_i}{h_i^{d+2}} g\left(\left\|\frac{y_j^t - x_i}{h_i}\right\|^2\right)}{\sum_{i \in KNN'(y_j^0)} \frac{1}{h_i^{d+2}} g\left(\left\|\frac{y_j^t - x_i}{h_i}\right\|^2\right)}.$$

5. Iterate steps 3 and 4 until convergence, $|y_j^{t+1} - y_j^t| < \delta$.

Clustering by Local Density Maxima

— Classic mean shift clusters points associated with the same stationary point together.

— Effective mean shift considers only effective neighbors and thus induces an approximation error as computing local density maxima.

For compensating the approximation error, we further merge registers with stationary points within a threshold distance ε into a cluster (ε is very small in our experiments).

.

Complexity Analysis

- Effective mean shift counts only effective neighbors by KNN and selects a proper bandwidth for each register, thus expediting the search of local density maxima.

Consider n registers, K effective neighbors for each register, T iterations to convergence, and C clusters generated.

Shifting to local density maxima can be done in $O(TKn)$ time, while register reassignment and cluster relocation can be done in $O(Cn)$ time. Hence, effective mean shift is of complexity $O(TKn+Cn)$,where $K \ll n$ and $C \ll n$.

.

Experimental Results

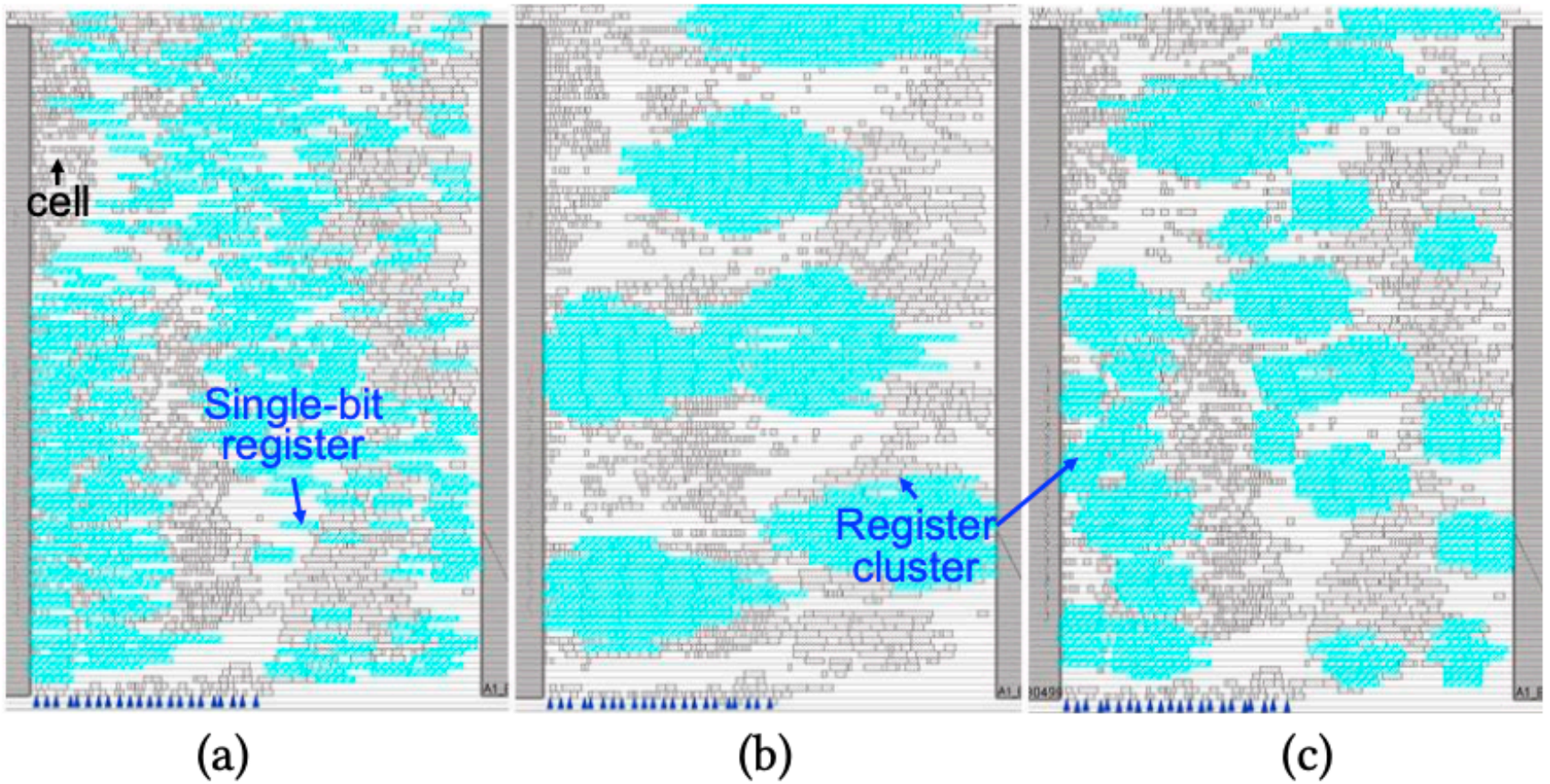


Figure 9. Partial layouts (superblue16). (a) Non-clustered. (b) Weighted K-means. (c) Effective mean shift.

Observations

It can be seen that weighted K-means tends to generate large clusters and induce large displacement for outliers, thus incurring significant placement disruption.

Table 3. Benchmark Statistics.

Circuit	# of Cells	# of Registers
superblue16	981,559	142,543
superblue18	768,068	101,758
superblue4	796,645	167,731
superblue5	1,086,888	110,941
superblue3	1,213,253	163,107
superblue1	1,209,716	137,560
superblue7	1,931,639	262,176
superblue10	1,876,130	231,747

Table 4. Pseudo Power of Multi-bit Register Library.

# of Bits	Normalized Pseudo Power per Bit
1	1.000
2~3	0.860
4~7	0.790
8~15	0.755
16~31	0.738
32~63	0.729
64~80	0.724

Used for power ratio — See next slide

Table 5. Comparison on Cluster Size, Displacement, and Runtime with Weighted K-Means [11].

Circuit	Method	Cluster Size		Displacement	Runtime (s)	
		Min	Max	Average	Para.	Seq.
superblue16	WK	34	80	56000.54	2370	
	Ours	1	55	22353.75	35	186
superblue18	WK	35	80	60843.50	6080	
	Ours	1	70	25792.54	25	138
superblue4	WK	34	80	48129.71	8470	
	Ours	1	56	19446.86	51	311
superblue5	WK	32	80	69453.46	3590	
	Ours	1	78	29747.90	28	131
superblue3	WK	28	80	54968.00	9098	
	Ours	1	79	25696.45	45	244
superblue1	WK	42	80	64158.15	5295	
	Ours	1	62	24456.03	40	200
superblue7	WK	39	80	54761.63	37692	
	Ours	1	79	26048.28	91	513
superblue10	WK	26	80	57643.75	27474	
	Ours	1	79	27914.53	75	412
Ratio	WK/Ours			2.33	215.03	39.42

Observations Contd..

Table 6. Comparison on Timing and Power with Weighted K-Means (WK) [11] and Non-Clustered Design (NC).

Circuit	Method	Timing			Power				
		WNS	TNS (ns)	TNS Degradation Ratio	Clock Routed WL (um)	Ratio	#Buffers	Ratio	Clock Sink Power Ratio
superblue16	NC	-6.2	-1532.0	0.00%	934,654	100.00%	3,414	100.00%	100.00%
	WK	-6.6	-2120.9	-38.44%	196,543	21.03%	1,872	54.83%	72.47%
	Ours	-6.2	-1629.8	-6.38%	214,560	22.96%	1,873	54.86%	74.86%
superblue18	NC	-9.1	-5148.3	0.00%	629,463	100.00%	2,449	100.00%	100.00%
	WK	-9.4	-5834.8	-13.33%	143,471	22.79%	1,314	53.65%	72.47%
	Ours	-9.1	-5250.0	-1.98%	144,009	22.88%	1,228	50.14%	74.32%
superblue4	NC	-9.7	-15669.9	0.00%	1,017,709	100.00%	4,303	100.00%	100.00%
	WK	-10.1	-16738.6	-6.82%	214,560	21.08%	2,124	49.36%	72.47%
	Ours	-9.9	-15830.8	-1.03%	234,966	23.09%	2,072	48.15%	74.91%
superblue5	NC	-30.2	-19866.8	0.00%	928,619	100.00%	3,626	100.00%	100.00%
	WK	-32.3	-20607.3	-3.73%	273,496	29.45%	2,251	62.08%	72.51%
	Ours	-30.3	-19898.6	-0.16%	291,267	31.37%	2,355	64.95%	74.16%
superblue3	NC	-18.9	-7892.9	0.00%	1,047,502	100.00%	4,251	100.00%	100.00%
	WK	-19.7	-8584.5	-8.76%	266,706	25.46%	2,054	48.32%	72.48%
	Ours	-18.9	-8106.1	-2.70%	262,588	25.07%	2,133	50.18%	74.14%
superblue1	NC	-10.2	-6778.5	0.00%	1,047,502	100.00%	3,759	100.00%	100.00%
	WK	-10.5	-7825.5	-15.45%	262,261	25.04%	2,052	54.59%	72.47%
	Ours	-10.2	-7334.7	-8.21%	255,708	24.41%	2,104	55.97%	74.87%
superblue7	NC	-19.4	-12531.2	0.00%	1,702,650	100.00%	6,482	100.00%	100.00%
	WK	-20.9	-13591.3	-8.46%	362,256	21.28%	3,427	52.87%	72.48%
	Ours	-19.2	-12757.0	-1.80%	379,577	22.29%	3,341	51.54%	74.31%
superblue10	NC	-48.7	-151000.0	0.00%	1,660,396	100.00%	6,189	100.00%	100.00%
	WK	-42.7	-139000.0	7.95%	379,246	22.84%	3,210	51.87%	72.48%
	Ours	-42.3	-141000.0	6.62%	408,500	24.60%	3,495	56.47%	74.25%
Average	NC			0.00%		100.00%		100.00%	100.00%
	WK			-10.88%		23.62%		53.45%	72.48%
	Ours			-1.95%		24.58%		54.03%	74.48%

Doubt

- Is it worth putting in so much effort for ~(9-10%) difference in TNS when compared with Classical Mean Shift? Agreed that the displacement of registers is addressed?

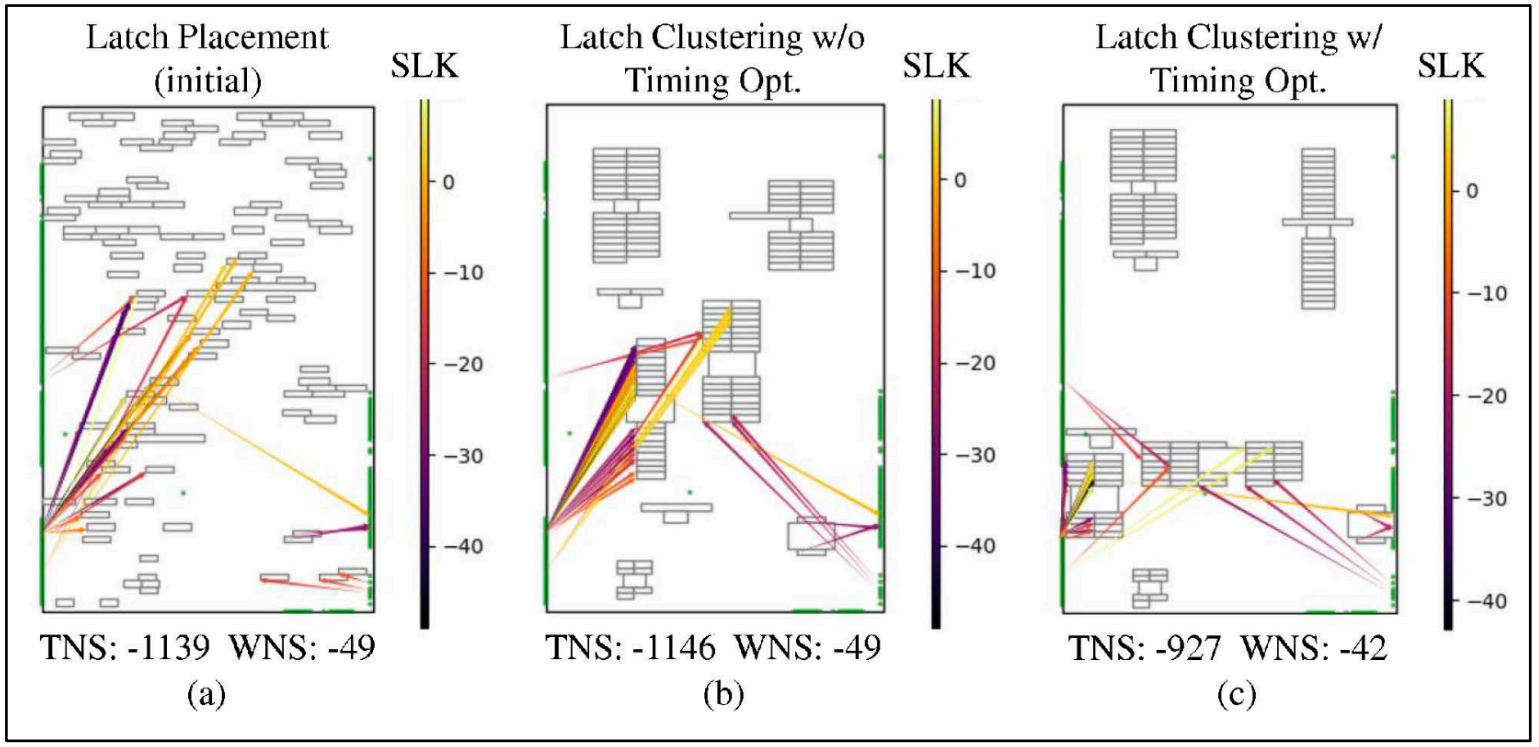
- Answer — Routing is easier. Detours are avoided. (Still exploring). Incremental research/ results are equally important. This probably works. TSMC has taken this approach. i.e incrementally shrinking the tech node.

Latch Clustering

- Most existing latch clustering researches mitigate timing disruptions by indirectly minimizing latch dis- placement during clustering, which is inaccurate and insufficient for timing closure in the design flow.
- Most researches do not control the amount of inserted clock buffers during clustering, which is the key factor to provide flexibility for timing and power trade-off.
- To address the two issues above, this paper presents a novel timing-power co-optimized latch clustering framework: we augment an integer linear programming (ILP) formulation of a facility-location allocation (FLA) problem to
- (1) directly optimize timing with a path-based timing model and
- (2) accurately control the number of inserted buffers by the FLA formulation for power optimization.
- Experimental results show 46% total negative slack timing overhead reduction, and 21% reduction for total power consumption.

Latch Clustering — Introduction

- Key circuit metrics for high-performance designs, such as timing and power, are particularly sensitive to latch position
- Main Idea — given an initial timing-optimized latch placement (see Figure (a)), neighboring latches are first clustered together, then placed near a driving local clock buffer (LCB) (see Figure (b))
- The clock skew requirements can be achieved effectively because clustered latches are placed closely to the corresponding driving LCBs and clock power can be saved by gating signals to LCB and not local leaf latches.
-



Latch Clustering — Challenges

- **1. Timing degradation after latch clustering should be minimized**. (The length reduction of clock interconnect comes at a cost of the data signal length increase, which harms the timing closure)
- **2. The second challenge is overlap removal between clusters during latch clustering.** (Latches in a high-performance design may be driven by different enable signals. Also because global placement may spread latches throughout the whole placeable chip area, latch clusters with optimized timing can inevitably be overlapped, leading to an infeasible latch placement solution.)
- **3. The third challenge is to satisfy electrical requirements when forming latch clusters.** (For a high-performance circuit, a steep clock slew rate is required, which needs a large driving strength for LCBs, and thus limits the number of latches driven by an LCB. Further, aligning pins of clustered latches can help reduce the local clock wirelength. and thus clock skew. Therefore, it is critical to satisfy the fanout limit constraints and pin alignment constraints when forming a latch cluster)
- **4. The last challenge is to trade off between timing and power optimizations during latch clustering.** (By splitting a timing-critical latch cluster (which contains a timing-critical latch) in two separate clusters can effectively improve timing. Nevertheless, the timing improvement comes at the cost of the increased number of LCBs, and thus increased clock power)
-
-

Facility Location Allocation Problem (ILP)

Problem Description:

Based on the class schedules, there exist different number of students in each building. We can use this information to estimate the demand per semester for coffee at each of the buildings.

Assume that the potential coffee shops in each building can meet only a certain amount of the total coffee demand. This represents the capacity of the coffee shop opened in building i.

⇨ Therefore, the demand points are the students in the buildings, and the supply points are the coffee shops in the buildings.

Each coffee shop can meet the demand of the students in all of the existing buildings at a cost, and opening a coffee shop in a building will also cost a fixed operating cost.

Sets:

F: set of buildings to locate a coffee shop in, indexed on i = 1,2,3,4.

D: set of buildings in which there exist students whose coffee demands are to be met, indexed on j = 1,2,3,4,5,6.

Parameters:

D_j: the total number of students in building *j* who buy coffee on campus.

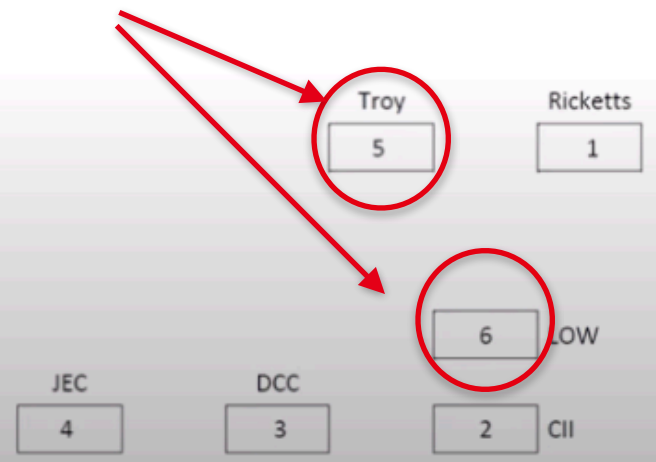
K_i : capacity of coffee shop that is to be opened in building *i* (the maximum number of students that the coffee shop in the building can serve).

c_{ij} : the cost, represented by the amount of time a student in building j can walk to building i in which there is a coffee shop.

More general: If the production costs vary based on buildings, then we could also include the cost of producing and shipping one unit from factory *i* to market *j*

f_i : fixed cost of operating coffee shop *i*

Small Buildings (Hence Ignored)



Supply Points — 4
Demand Points — 6 (Because there are students in all the buildings)

We want to determine which coffee shops to open, as well as the best allocation of students to the opened coffee shops that minimizes the total cost.

FLA Problem Contd..

Objective Function [Minimize Total Cost (Fixed Cost + Cost due to walking)]

Decision Variables:

y_i : 1 if a coffee shop in building *i* is opened, and 0 otherwise.

x_{ij}: number of students that go to building j from building i to get their coffee.

Cost due to Walking —>

C_{ij}	1	2	3	4	5	6
1	25	180	230	300	40	120
2	180	25	40	110	180	40
3	230	40	25	40	180	80
4	300	110	40	25	180	120

Fixed Cost —>

F, fixed cost
5000
7500
9000
10000

Important Aspects of the Problem
1. Data Parameters
2. Decision Variables
3. Objective Function
4. Constraints

FLA Problem

- **Fixed Cost — Min[5000*y1 + 7500*y2 + 9000*y3 + 10000*y4]**
- **y — 1 [open] & 0[closed]**

- **Variable Cost — [25*x11 + 180*x12 + 230*x13 … 180*x45 + 120*x46]**
- **x[ij] — No. Of students going from building ‘j’ to building ‘i’ to get their coffee**

- **Idea — Minimise Fixed + Variable Cost.**

Constraints

- 1. Demand of each customer must be met
- 2. Coffee shops cannot server more customers than their capacities.

j	D, demand
1	100
2	90
3	125
4	150
5	80
6	75

i	K, capacity
1	250
2	200
3	350
4	450

Decision Variables:

y_i : 1 if a coffee shop in building *i* is opened, and 0 otherwise.

x_{ij}: number of students that go to building j from building i to get their coffee.

Constraints

- Constraint 1 — Where can the student in building 1 get coffee from? (Ans —All 4 coffee shops)
- **[x11 + x21 + x31 + x41 = 100]**
- **[x12 + x22 + x32 + x42 = 90] .. and so on**

- Constraint 2 — To many students in the University can I serve coffee (Ans — Students from all buildings)
- **[x11 + x12 + x13 + x14 + x15 + x16 <= 250*y1] (y1 — indicates open[1] or closed[0])**

- **DECISION VARIABLES —**
- x[ij] >=0 for all j
- y[i] {0,1} for all i

General Form

Min 5000y1 +7500y2 +9000y3 + 10000y4 + 25x11 + 180x12 + ...
+ 180x45 + 120x46

s.t.

x11 + x21 + x31 + x41 = 100

x12 + x22 + x32 + x42 = 90

x13 + x23 + x33 + x43 = 125

x14 + x24 + x34 + x44 = 150

x15 + x25 + x35 +x45 = 80

x16 + x26 + x36 + x46 = 75

x11 + x12 + x13 + x14 + x15 + x16 <= 250y1

x21 + x22 + x23 + x24 + x25 + x26 <= 200y2

x31 + x32 + x33 + x34 + x35 + x36 <= 350y3

x41 + x42 + x43 + x44 + x45 + x46 <= 450y4

x_{ij} >= 0 for all i = 1,2,3,4; j=1,2,3,4,5,6

y_i is binary for all i = 1,2,3,4

Objective:

$$Min \sum_{i=1}^{n} f_i y_i + \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = D_j \quad \text{for} \quad j = 1,\ldots, m$$

$$\sum_{j=1}^{m} x_{ij} \le K_i \, y_i \quad \text{for} \quad i = 1,\ldots, n$$

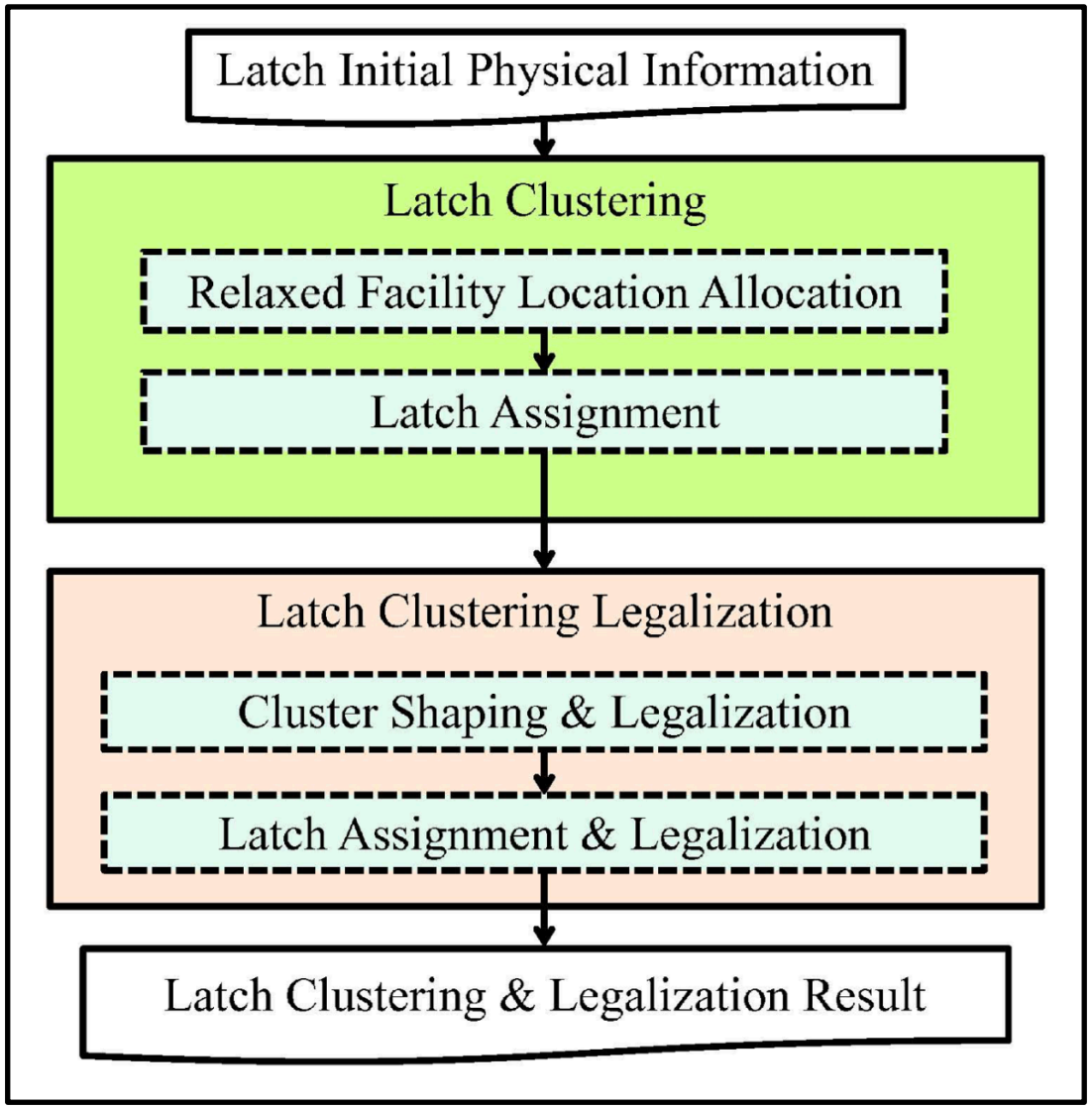
$$x_{ij} \ge 0 \quad for \; i = 1,\ldots, n, \text{ for } \quad j = 1,\ldots, m$$

$$y_i = \{0,1\} \quad \text{for} \quad i = 1,\ldots, n$$

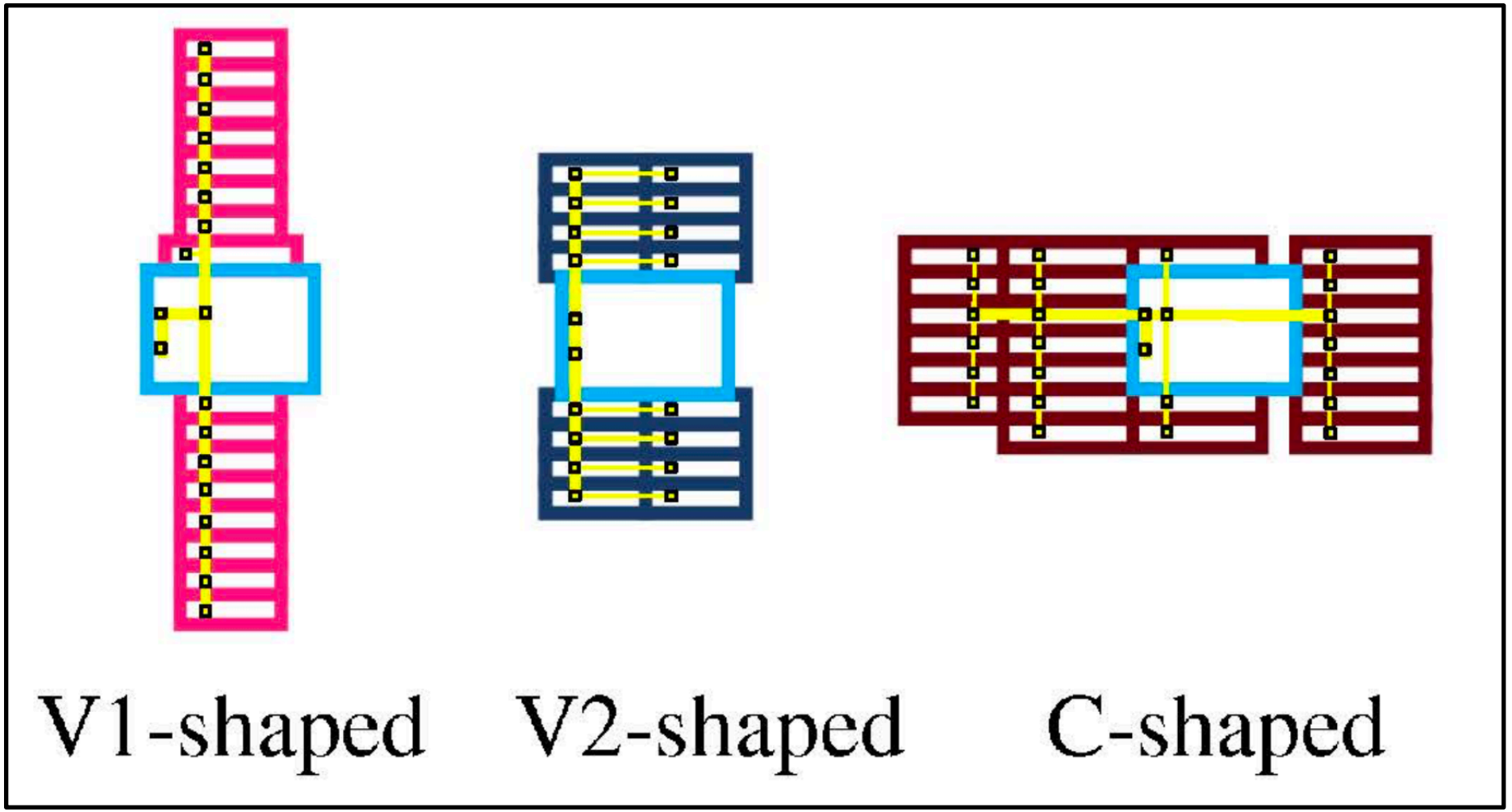
Latch Clustering (Goals)

- Given an initial latch placement and a target slack threshold SLKth, form a set of K latch clusters to (1) maximize iVNS and TNS, and (2) minimize the maximum and total latch displacements, while satisfying the cluster size constraints with size_limit.
- size_limit is a given constant value, and denotes the maximum fanout number of an LCB. disp_limit; is a given maximum displace- ment value for latch *i,* according to its timing criticality. rmm_limit is also a given constant value, and denotes the maximum number of clusters.

Program Flow —



Latch Clustering Legalisation



Things I did not understand

- Timing Optimisation in Latch Clustering.
- Maximising the WNS/TNS as one of the cost functions.
- Maximising slack on the critical paths.
- ILP reduction on Latch Clustering.

Latch Clusters

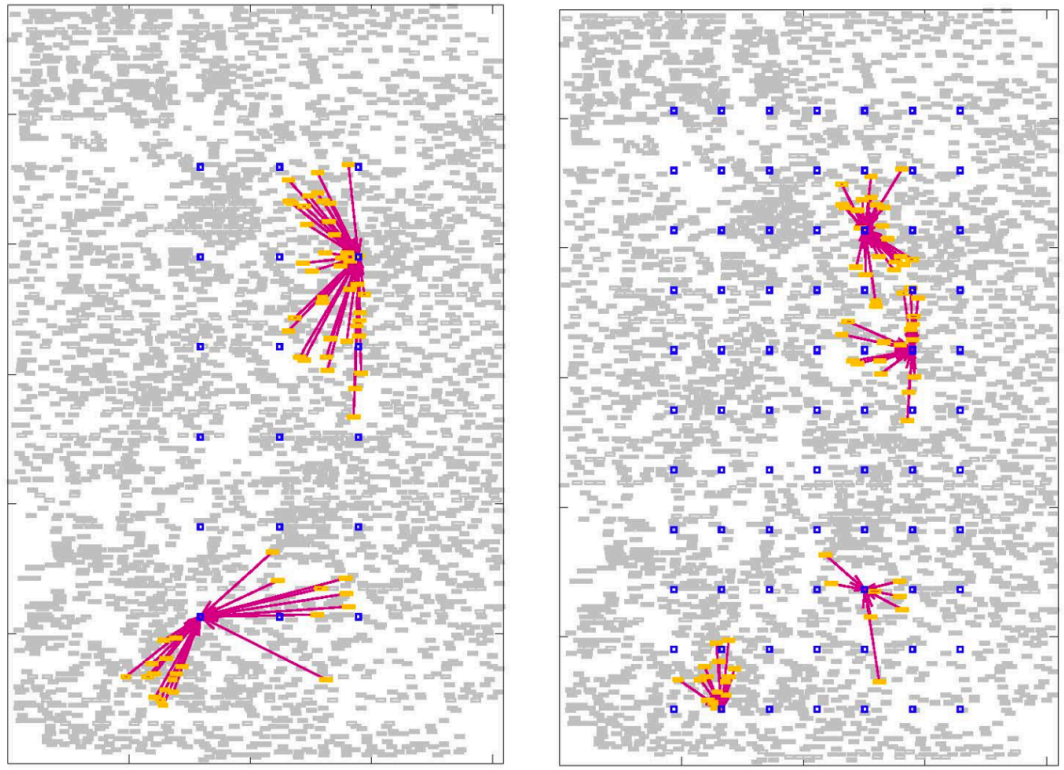
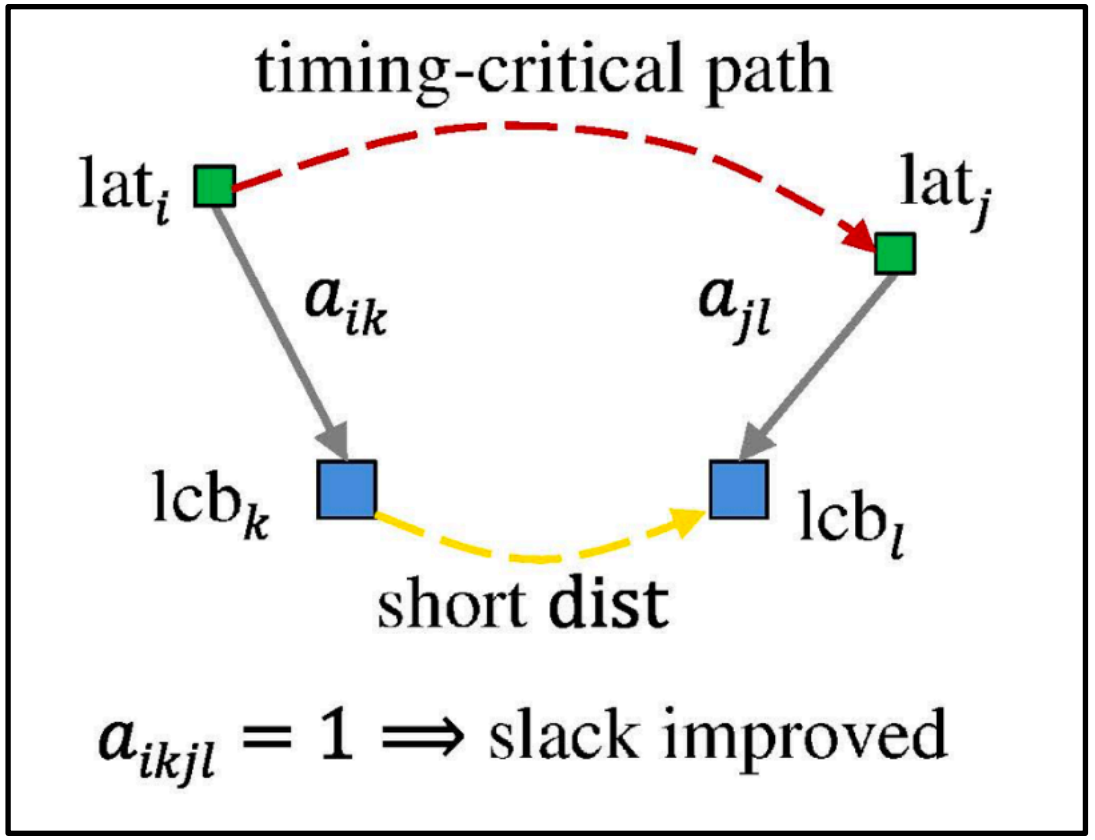


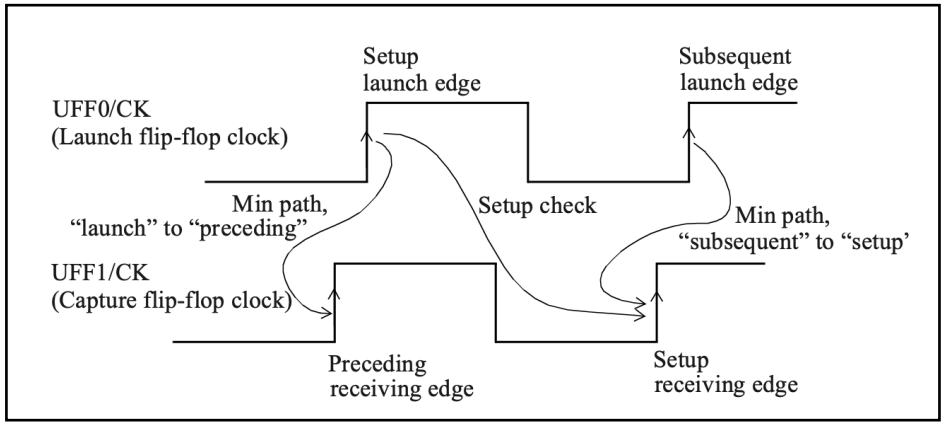
Fig. 6: FLA clustering results with (a) a minimized number of clusters (b) a larger number of clusters with a smaller spacing for the potential cluster positions (in blue).

Slack Improvement

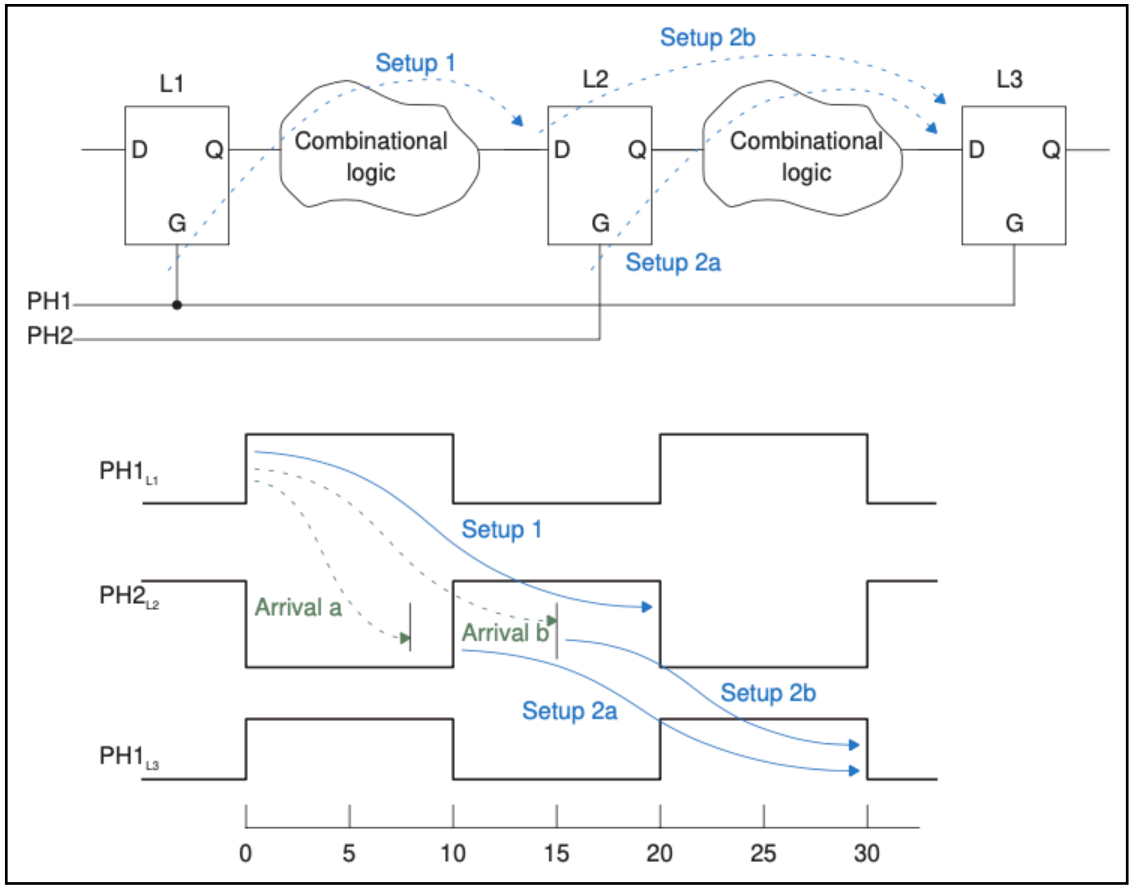


Latch Timing

- **Latch timing** is much much harder than Flip-Flop Timing Analysis majorly because it is Level Triggered and not edge triggered. Only **Intel** has cracked this. They make chips (server/gaming) and they managed to push frequencies upto 5GHz. They in-fact have their own **Timing Analyzer** for closing the Timing. They do not use PrimeTime for timing closure.

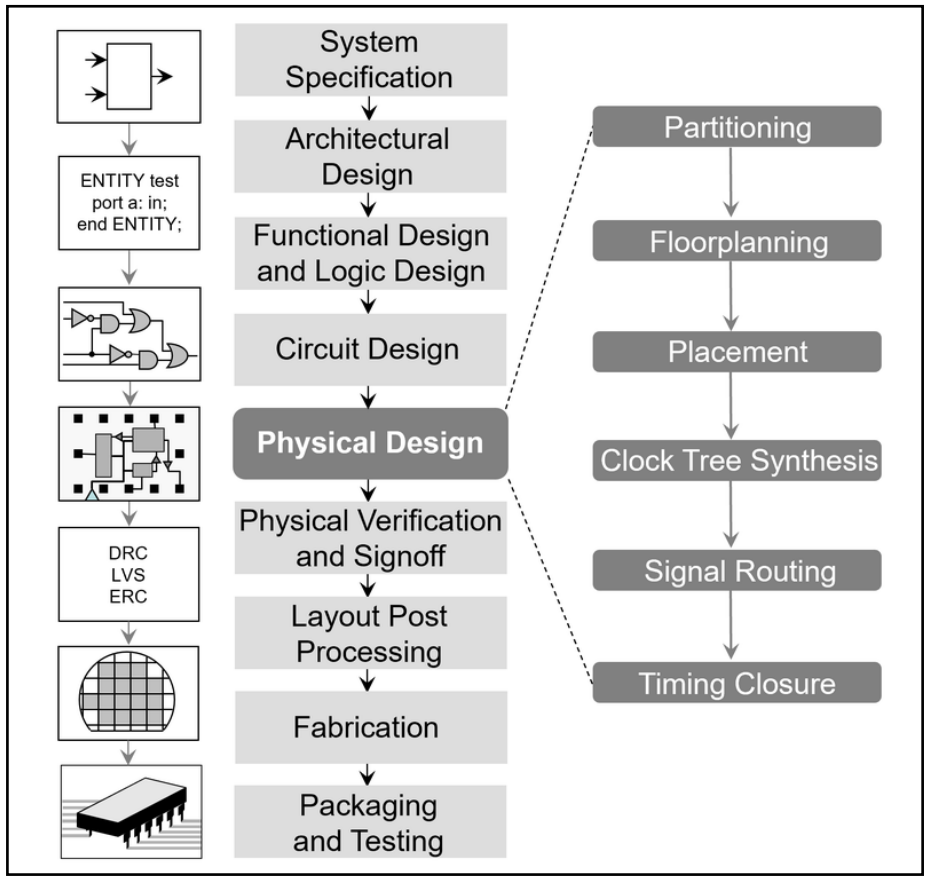


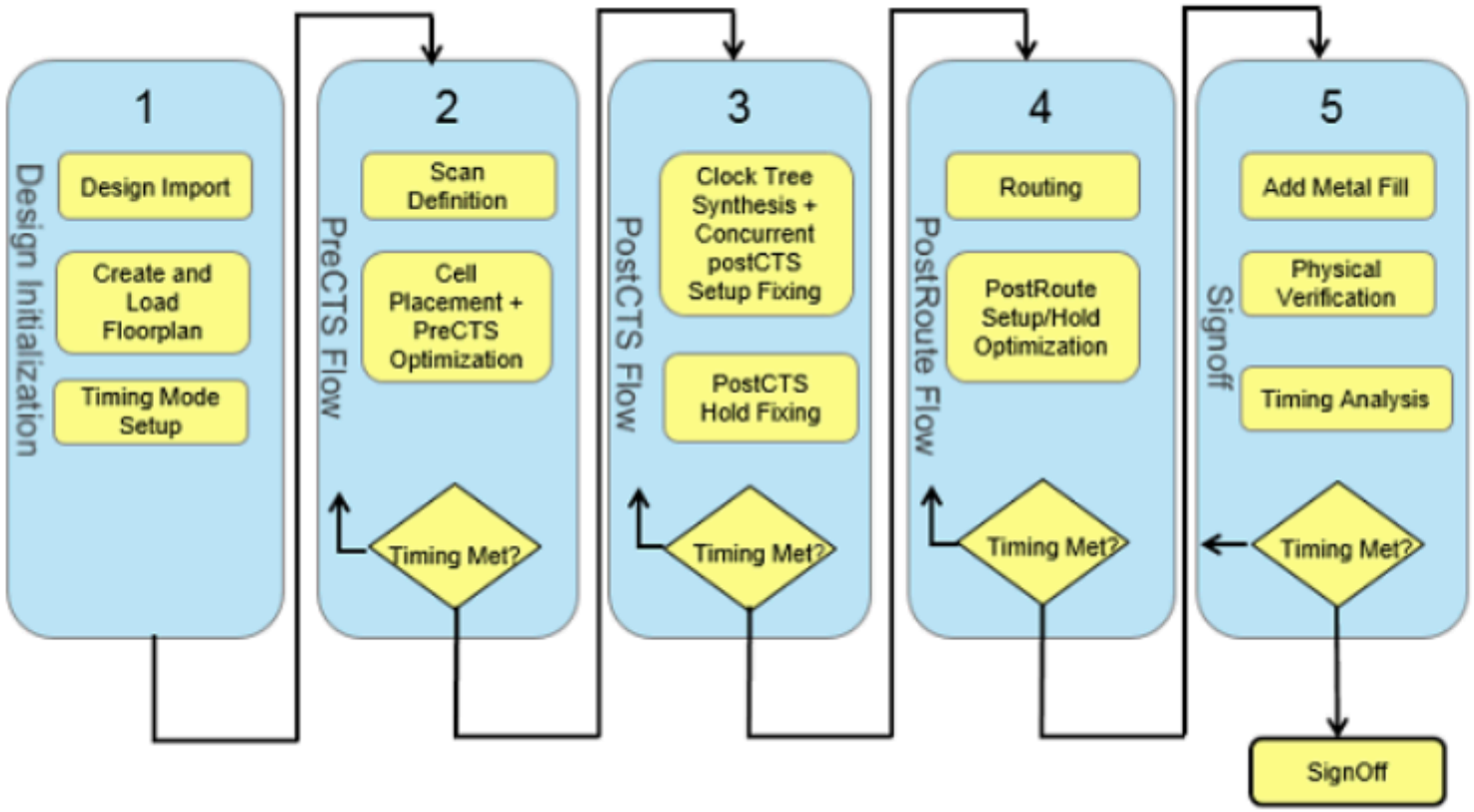
Flip Flop Timing Analysis



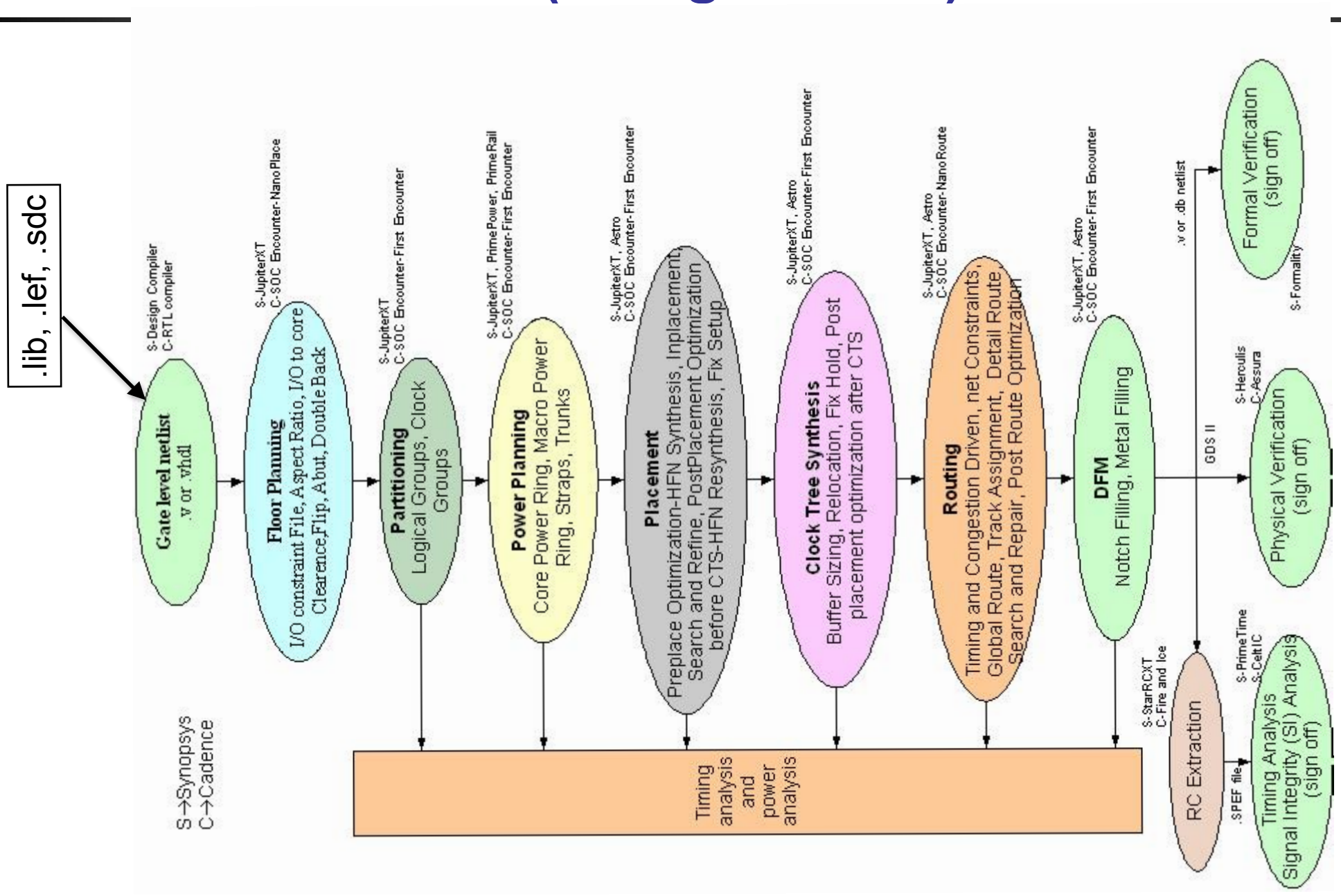
Latch Timing Analysis

Appendix — Physical Design Flow (Designer PoV)

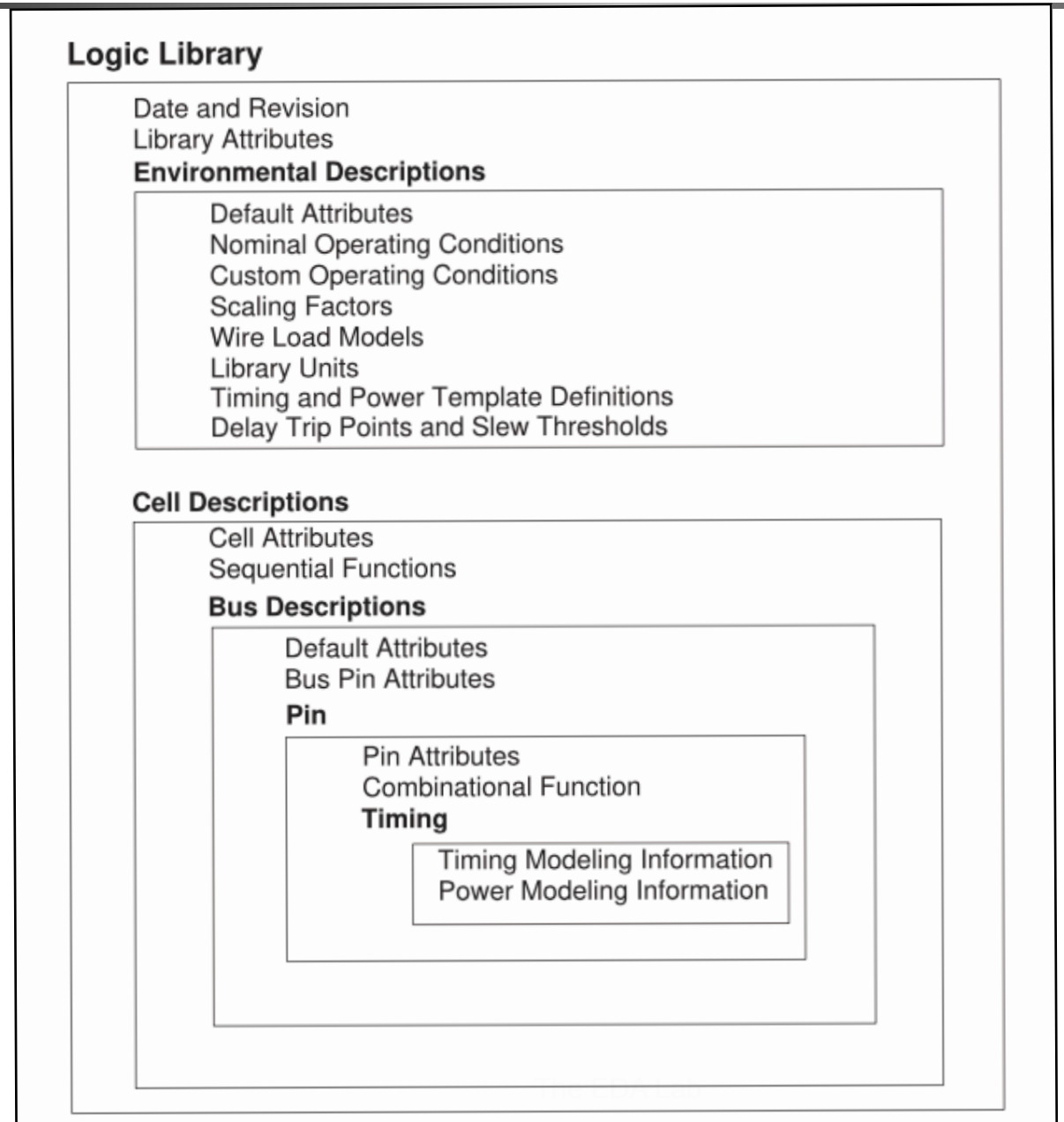




PD Flow (Designer PoV) contd..



Structure of a Logic Library (.lib)



Basic SDC File (Synopsys Design Constraints)

```
create_clock -period 20 -waveform {0 6} -name SYS_CLK  [get_ports SYS_CLK]
create_generated_clock -divide_by 2 -source [get_ports sys_clk] -name gen_sys_clk [get_pins UFF/Q]
set_clock_latency 1.86 [get_clocks clk250]
set_clock_uncertainty -setup -rise -fall 0.2 [get_clocks CLK2]
set_clock_transition -min 0.5 [get_clocks SERDES_CLK]
set_clock_transition -max 1.5 [get_clocks SERDES_CLK]
set_input_delay -clock virtual_mclk 2.5 [all_inputs]
set_output_delay 0.40 [all_outputs]
set_max_delay -from [get_clocks FIFOCLK] -to [get_clocks MAINCLK] 3.5
```

Placement Tool Flow

1. **Initial placement (initial_place)**
   During this stage, the tool merges the clock-gating logic and performs coarse placement. If a block contains scan chains that are annotated by reading a SCANDEF file, the tool also performs scan chain optimization.

2. **Initial DRC violation fixing (initial_drc)**
   During this stage, the tool removes existing buffer trees and performs high-fanout-net synthesis and electrical DRC violation fixing.

3. **Initial optimization (initial_opto)**
   During this stage, the tool performs timing, area, congestion, and leakage-power optimization.

4. **Final placement (final_place)**
   During this stage, the tool performs incremental placement to improve timing and congestion, and legalizes the design.

5. **Final optimization (final_opto)**
   During this stage, the tool performs further optimization and legalization to improve timing and congestion.

CTS Tool Flow

The clock_opt command consist of the following stages:

1. The **build_clock** stage, during which the tool synthesizes and optimizes the clock trees for all clocks in all modes of all active scenarios. After clock tree synthesis, the tool sets the synthesized clocks as propagated.

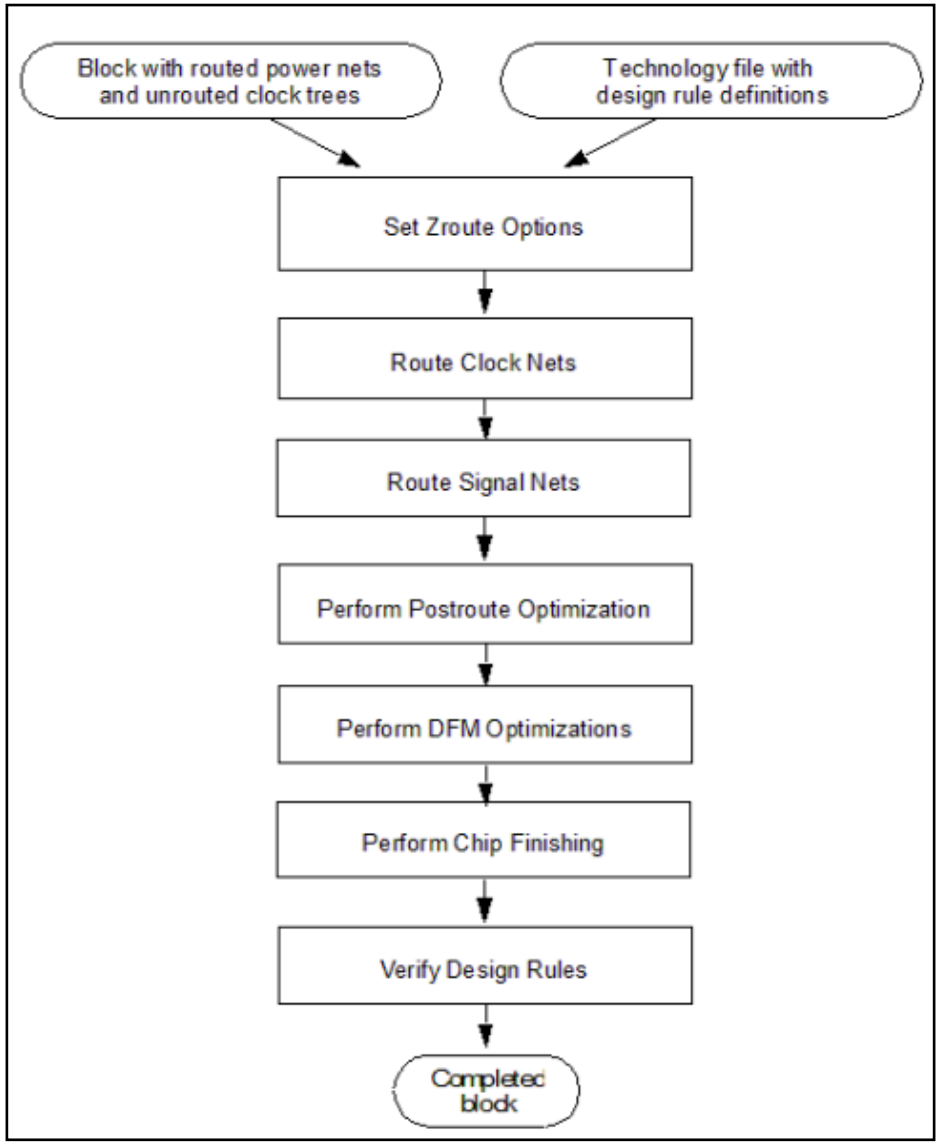
2. The **route_clock** stage, during which the tool detail routes the synthesized clock nets.

3. The **final_opto** stage, during which the tool performs further optimization, timing- driven placement, and legalization. It then global routes the block and performs extensive global-route-based optimization, which includes incremental legalization and route patching.

Routing Tool Flow

5 stage routing

1. Global Routing
2. Track Assignment
3. Detail Routing
4. ECO Routing
5. Routing Verification



Floorplanning Checks

1. Check PG connections (For macros & pre-placed cells only)
2. LP / MV checks on floorplan database
3. Check the power connections to all Macros, specially analog/special macros if any
4. All the macros should be placed at the boundary
5. There should not be any notches / thin channels. If unavoidable, proper blockages has to be added
6. Remove all unnecessary placement blockages & routing blockages (which might be put during floor-plan & pre-placing)
7. Check power connection to power switches
8. Check power mesh in different voltage area voltage area
9. Check pin-layers & check layer directions (H-V-H)

Placement — Goals + Checks

Different tasks in placement are listed below →

1. Pre-placement
2. Initial placement (Coarse placement)
3. Legalizations
4. Removing existing buffer trees
5. High Fan-out Net Synthesis (HFNS)
6. Iterations of timing/power optimizations
   [cell sizing, moving, net spitting, gate cloning,
   buffer insertion, area recovery]
7. Area recovery
8. Scan-chain re-ordering
9. TIE cell insertions

PLACEMENT GOALS

1. Timing, Power and Area optimizations
2. Routable design (minimal global & local congestion)
3. No/minimal cell density, pin density & congestion hot-spots
4. Minimal timing DRCs

Placement — Goals + Checks

Pre-Placement Checklist →

1. Check for any missing / extra placement & routing blockages
2. Don't use cell list & whether it is properly applied in the tool
3. Don't touch on cells & nets (make sure that, these are applied)
4. Better to limit the local density (Otherwise local congestion can create issue in routing / eco stages)
5. Understand all optimization options & placement switches set in the tool
6. There should not be any high WNS timing violations
7. Make sure that clock is set to ideal network
8. Take care of integration guidelines of any special IPs (These won't be reported in any of the checks). Have custom scripts to check these guidelines
9. Fix all the hard macros & pre-placed cells
10. Check the pin access

Post Placement Checklist →

1. Logical equivalence check & low power checks
2. Check legalization
3. Check PG connections of all the cells
4. Check congestion, place density & pin density maps. All these should be under control
5. Timing QoR / Convergence. There should not be any high WNS violations & TNS, NVP must be under control
6. Minimal max tran & max cap violations
7. Check whether all don't touch cells & nets are preserved
8. Check for don't use cells (Should be Zero/ same as post Syn)

Clock N/W Synthesis — Goals + Checks

Sanity checks to be done before CTS →

- Check legality.
- Check power stripes, standard cell rails & also verify PG connections.
- Timing QoR (setup should be under control).
- Timing DRVs.
- High Fanout nets (like scan enable / any static signal).
- Congestion (running CTS on congested design / design with congestion hotspots can create more congestion & other issues (noise / IR)).
- Remove don't_use attribute on clock buffers & inverters.
- Check whether all pre-existing cells in clock path are balanced cells (CK* cells).
- Check & qualify don't_touch, don't size attributes on clock components.

Checks to be done after CTS →

- Perform RC extraction of the clock nets and compute accurate clock arrival time.
- Adjust the I/O timings.
  - After implementing the clock tree, the tool can update the input and output delays to reflect the actual clock arrival time.
- Perform power optimization.
  - Use a large/Max clock gating fanout during insertion of the ICG cells.
  - Merge ICG cells that have the same enable signal.
  - Perform power-aware placement of ICG and registers.
- Check and fix any congestion hotspots.
- Optimize the scan chain.
- Fix the placement of the clock tree buffers and inverters.
- Perform placement and timing opt.
- Check for major hold time violation.

Routing — Goals & Checks

Goals of Routing

- Minimize the total interconnect/wire length.
- Maximize the probability that the tool can complete routing.
- Minimize the critical path delay.
- Minimize the number of layer changes that the connections have to make (minimizing the number vias).
- Complete the connections without increasing the total area of the block.
- Meeting the Timing DRCs and obtaining a good Timing QoR.
- Minimizing the congestion hotspots.
- SI driven: reduction in cross-talk noise and delta delays.

Pre-Routing Checks →

- Check if the ports of the standard cells are blocked i.e. the physical pins are not accessible.
- Checks for overlapping cells in the design. Overlapping causes pins to short and cause metal DRC violations.
- Check for pins underneath PG routes (they may be inaccessible and cause violations on metals) .
- Check if the ports of the top-level or macro cell are blocked and physically inaccessible.
- Check for pins that are outside the design boundary (Out-of-Boundary pins).
- Check for blocked PG ports.
- Check if there are frozen nets blocking ports.
- Check for blocked unconnected pins.
- Check if all pins in the design are on the routing tracks.

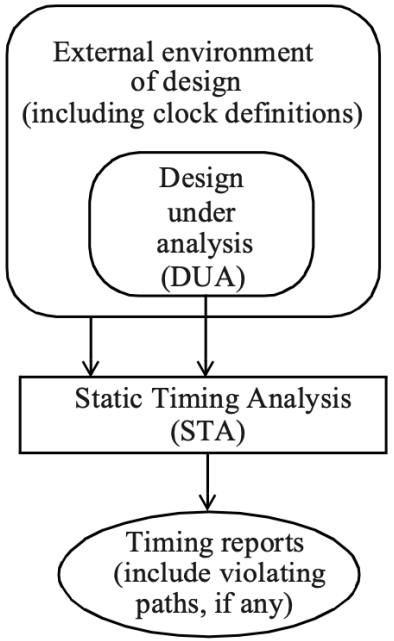
Post Route Optimization

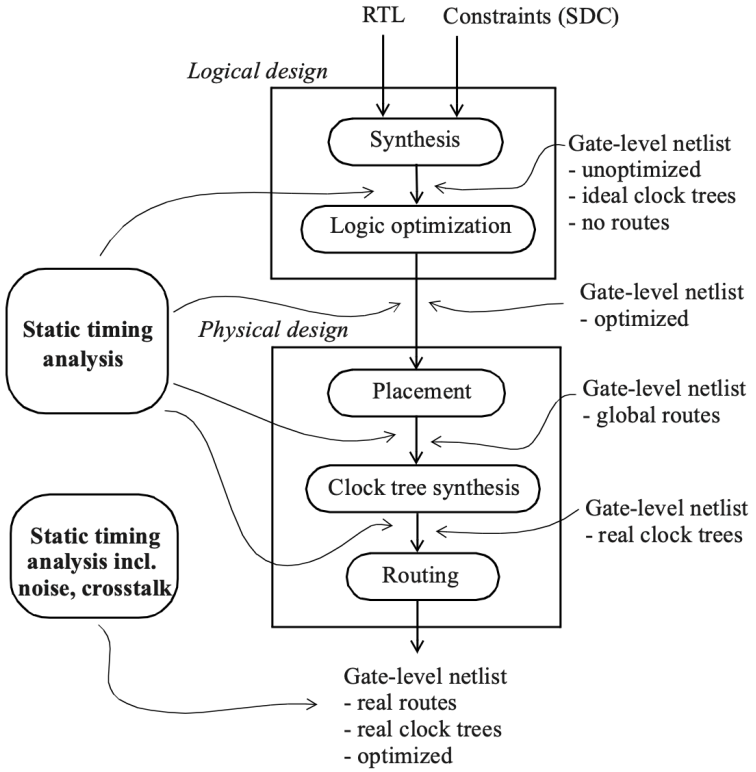
Pre-Routing Checks →

- Fixing timing violations.
- Fixing LVS (opens & shorts).
- Fixing DRCs.
- Fixing Timing DRCs (Meet max transition, max capacitance and max fanout).
- Finding & Fixing Antenna violations (using jumpers and antenna diodes).
- Area and Leakage power recovery.
- Fixing SI related issues.
- Redundant via insertion.

Appendix — STA Basics

— Static timing analysis (STA) is a method of validating the timing performance of a design by checking all possible paths for timing violations.
— STA breaks a design down into timing paths, calculates the signal propagation delay along each path, and checks for violations of timing constraints inside the design and at the input/output interface.

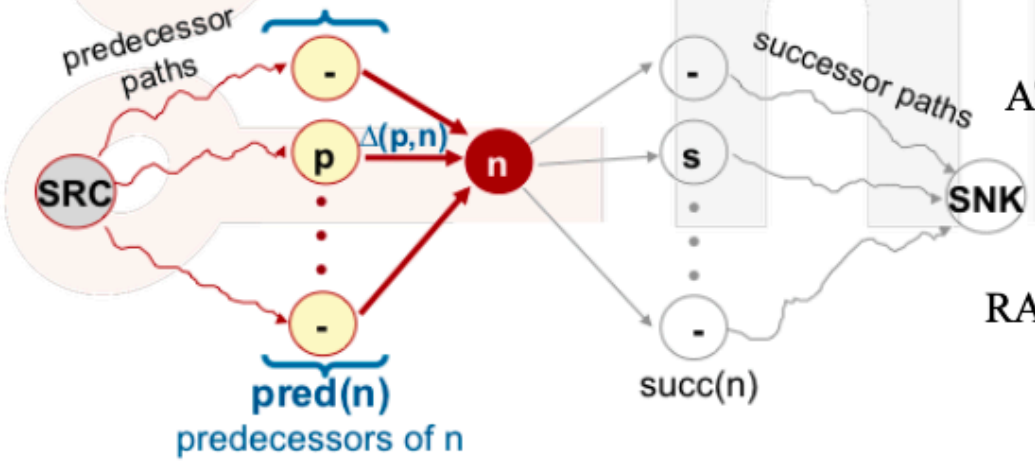




Arrival Time and Required Time Calculation

How do we compute ATs and RATs?

- **Recursively!**
  - The **Arrival Time** at a node is just the maximum of the ATs at the *predecessor* nodes plus the *delay from that node*.
  - The **Required Arrival Time** to a node is just the minimum of the RATs at the *successor* nodes minus the *delay to that node*.

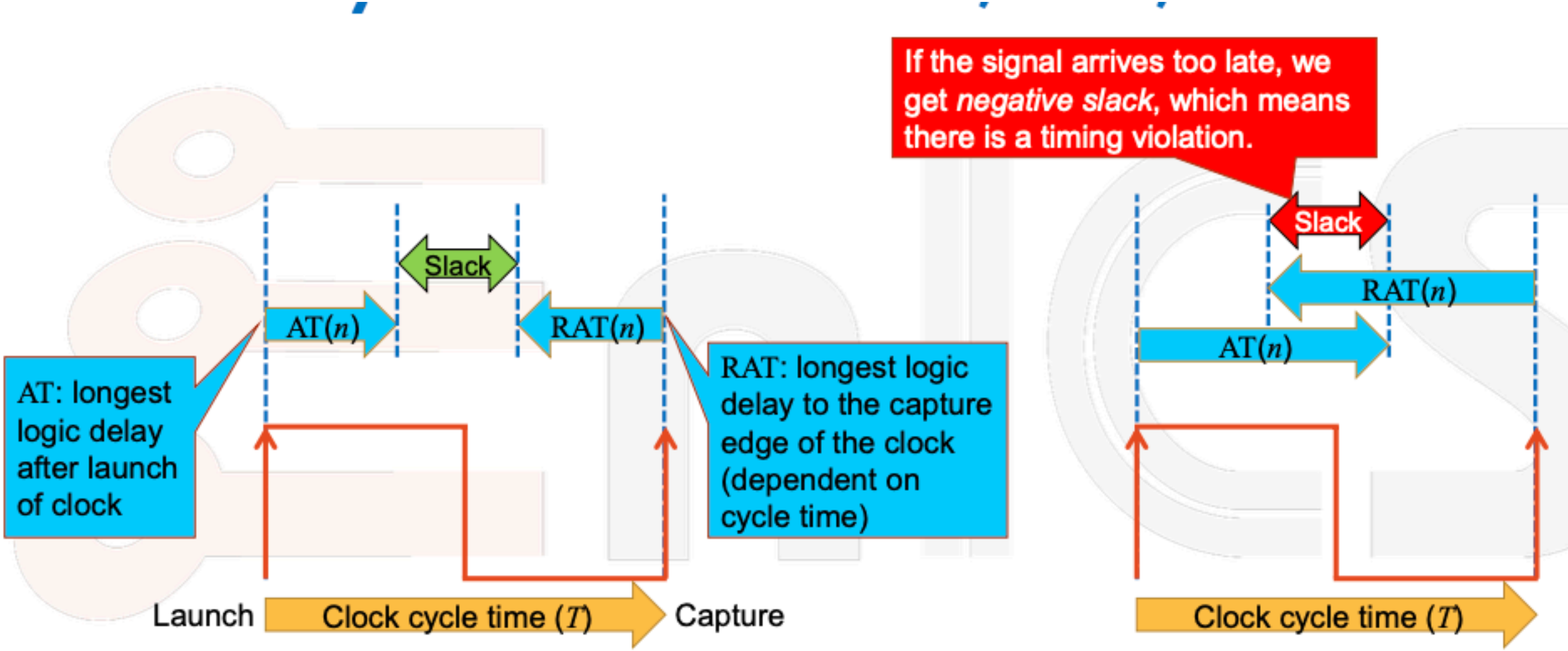


$$AT(n) = \begin{cases} 0 & n = \text{SRC} \\ \max\limits_{p \in \text{pred}(n)} \left[AT(p) + \Delta(p,n) \right] & n \neq \text{SRC} \end{cases}$$

$$RAT(n) = \begin{cases} T & n = \text{SNK} \\ \min\limits_{s \in \text{succ}(n)} \left[RAT(s) - \Delta(n,s) \right] & n \neq \text{SNK} \end{cases}$$

More Info — https://www.youtube.com/playlist?list=PLZU5hLL_713x0_AV_rVbay0pWmED7992G

Arrival Time and Required Time Calculation

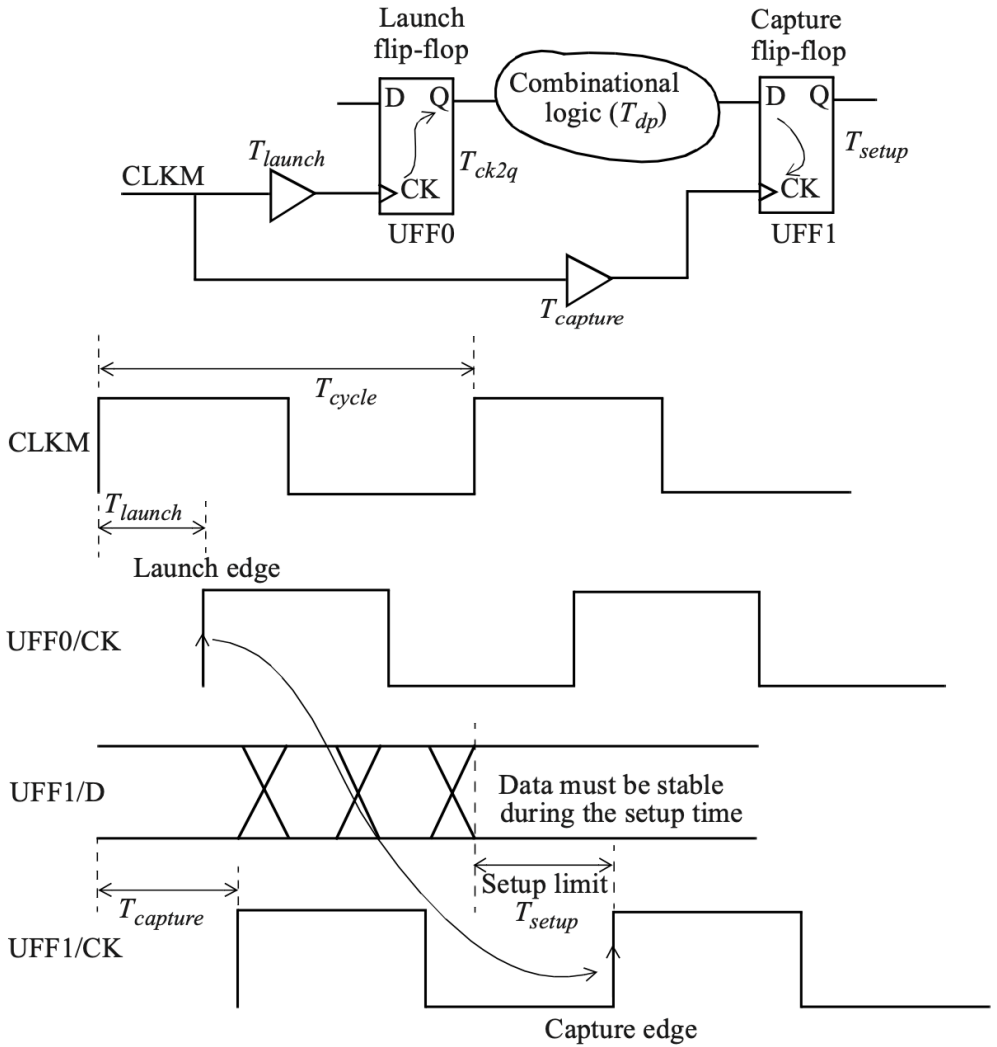


More Info — https://www.eng.biu.ac.il/temanad/files/2018/12/Lecture-5-STA.pdf

SETUP & HOLD CHECKS

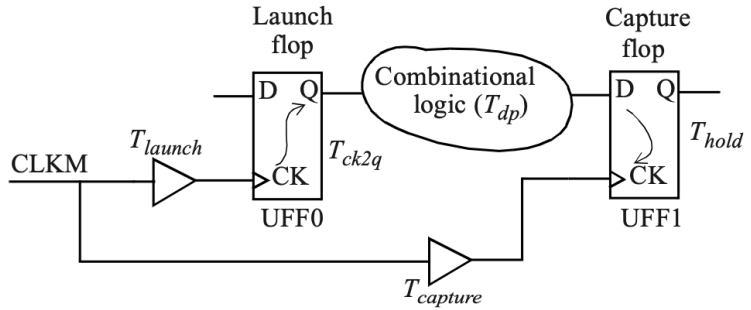
- **SETUP CHECK** — A setup check ensures that the data can arrive at a flip-flop within the given clock period.

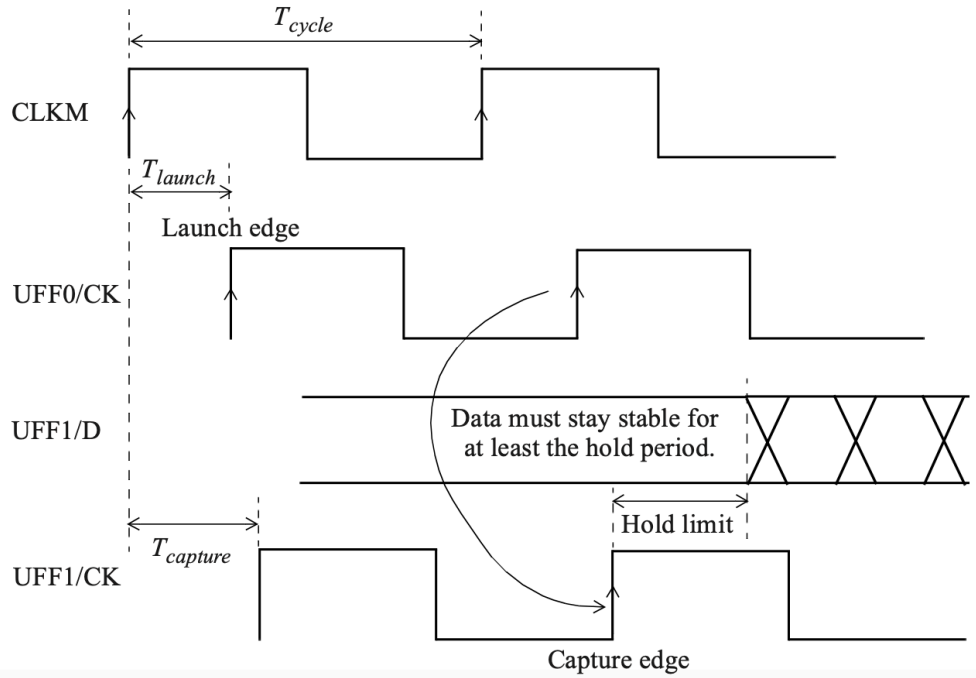
- ***T(launch) +T(clk2q) + T(dp) < T(cycle) + T(capture) - T (setup)***



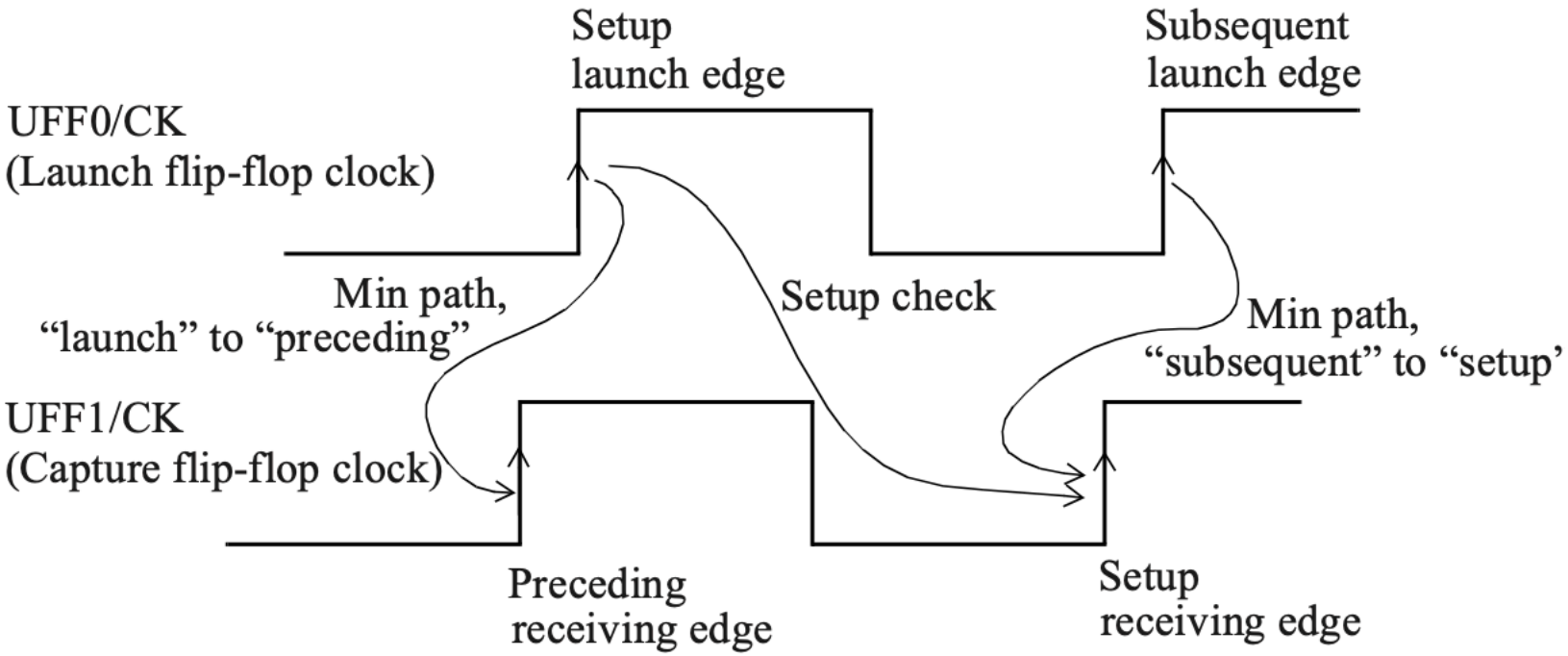
HOLD CHECKS

- **HOLD CHECK** — A hold check ensures that the data is held for at least a minimum time so that there is no unexpected pass-through of data through a flip-flop: that is, it ensures that a flip-flop captures the intended data correctly.
- **(L+1 —> C or L —> C-1) (L- launch, C- capture)**
- ***T(launch) +T(clk2q) + T(dp) > T(capture) +T(hold)***





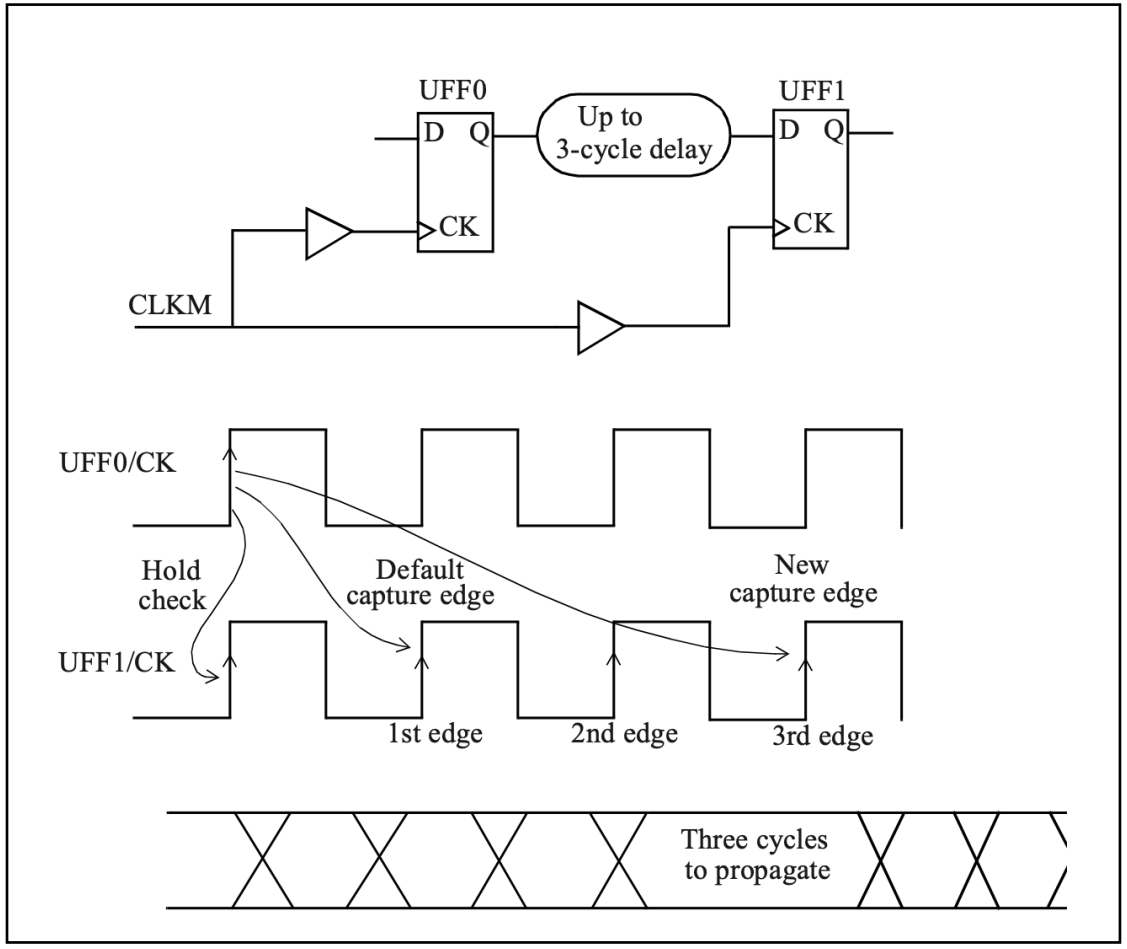
HOLD CHECKS contd.



MultiCycle Paths (SU Check)

In some cases, the combinational data path between two flip-flops can take more than one clock cycle to propagate through the logic. In such cases, the combinational path is declared as a multi-cycle path. Even though the data is being captured by the capture flip-flop on every clock edge, we direct STA that the relevant capture edge occurs after the specified number of clock cycles.

```
create_clock -name CLKM -period 10 [get_ports CLKM]
set_multicycle_path 3 -setup \
  -from [get_pins UFF0/Q] \
  -to [get_pins UFF1/D]
```

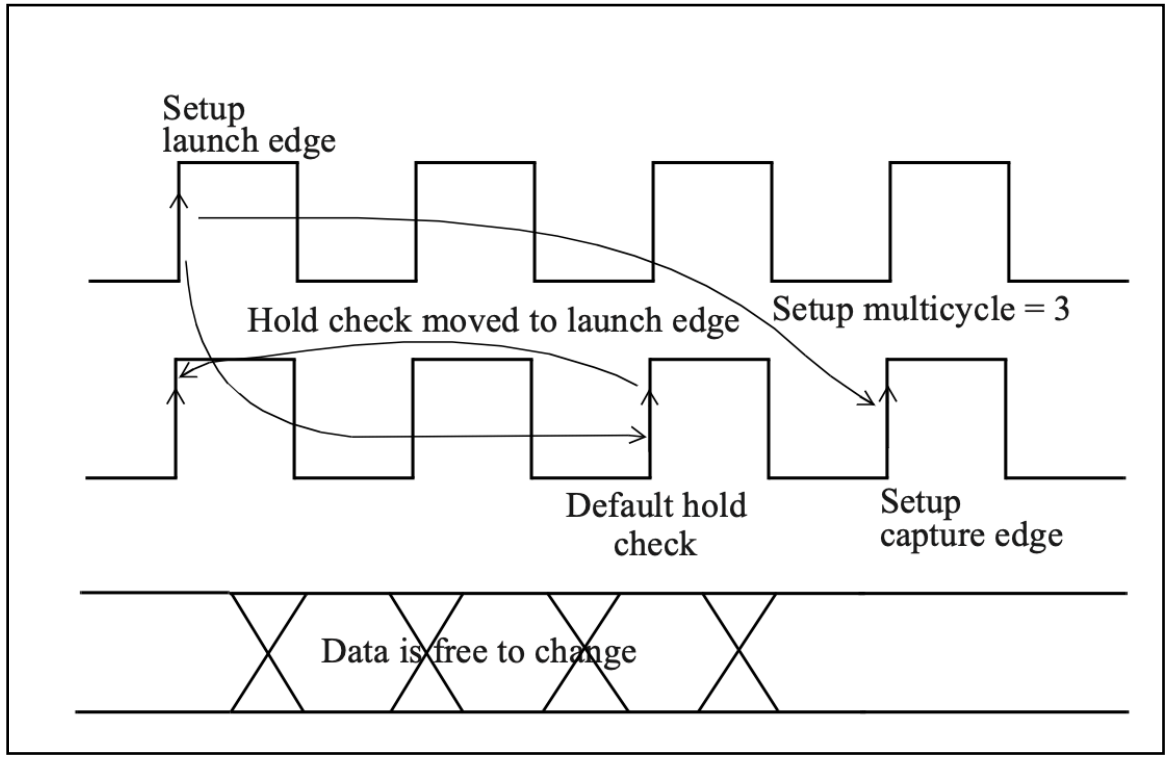


MultiCycle Paths (Hold Check)

In most common scenarios, we would want the hold check to stay as it was in a single cycle setup case. This ensures that the data is free to change anywhere in between the three clock cycles. A hold multicycle of two is specified to get the same behavior of a hold check as in a single cycle setup case. This is because in the absence of such a hold multicycle specification, the default hold check is done on the active edge prior to the setup capture edge which is not the intent. We need to move the hold check two cycles prior to the default hold check edge and hence a hold multicycle of two is specified. With the multicycle hold, the min delay for the data path can be less than one clock cycle.

```
set_multicycle_path 2 -hold -from [get_pins UFF0/Q] \
  -to [get_pins UFF1/D]
```

.



False Paths

It is possible that certain timing paths are not real (or not possible) in the actual functional operation of the design. Such paths can be turned off during STA by setting these as false paths. A false path is ignored by the STA for analysis.

Examples —

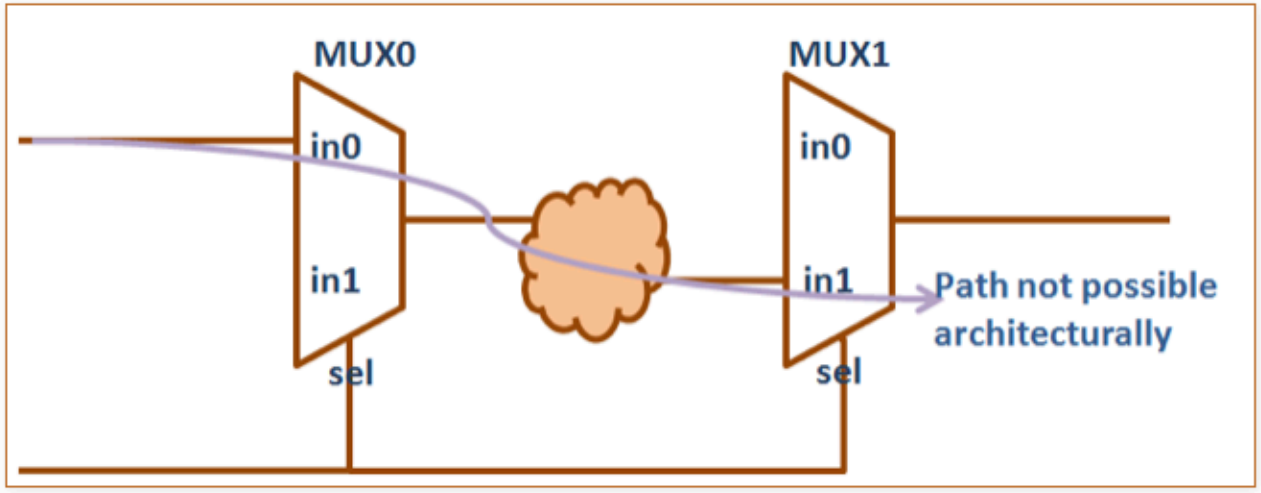
1. Path from one clock domain to another clock domain

2. Clock Pin of a flip-flop to input pin of another flip-flop

3. Architectural false paths (given in the figure)

```
set_false_path -from [get_clocks SCAN_CLK] \
  -to [get_clocks CORE_CLK]
# Any path starting from the SCAN_CLK domain to the
# CORE_CLK domain is a false path.
```

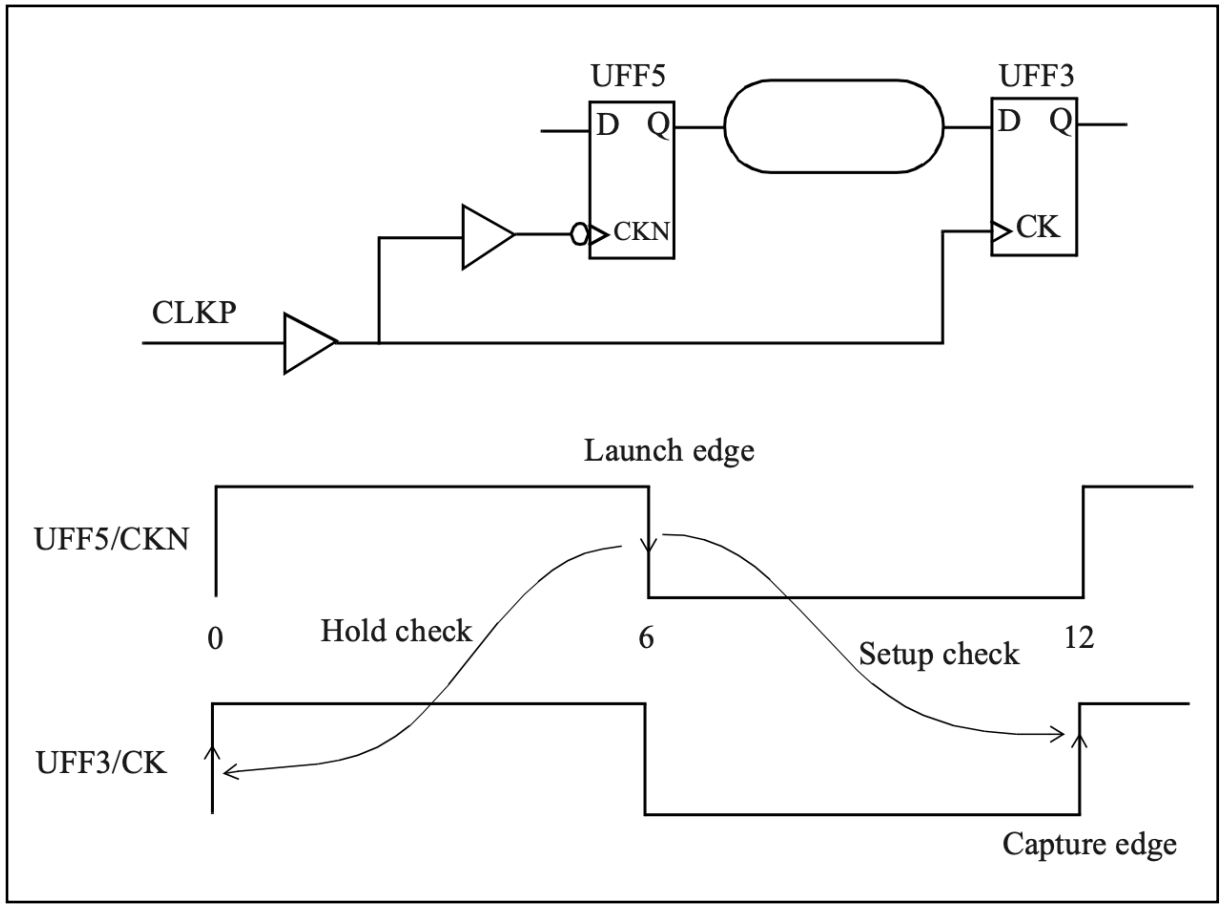
.



Half Cycle Paths

If a design has both negative-edge triggered flip-flops (active clock edge is falling edge) and positive-edge triggered flip-flops (active clock edge is ris- ing edge), it is likely that half-cycle paths exist in the design. A half-cycle path could be from a rising edge flip-flop to a falling edge flip-flop, or vice versa. The hold check always occurs one cycle prior to the capture edge.

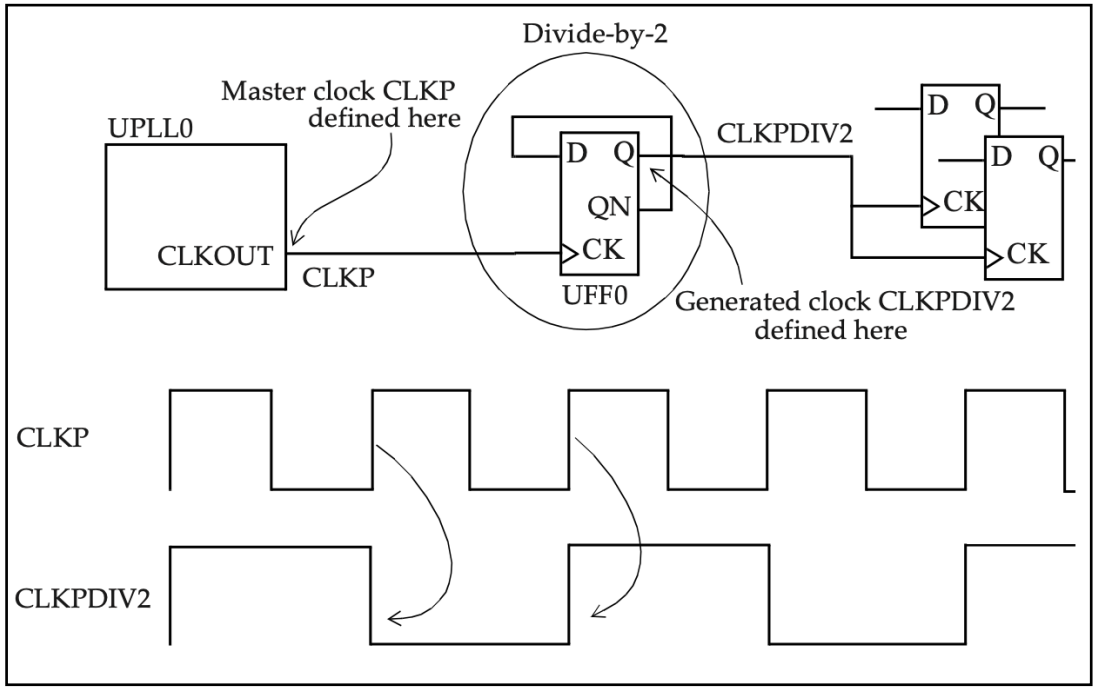
.



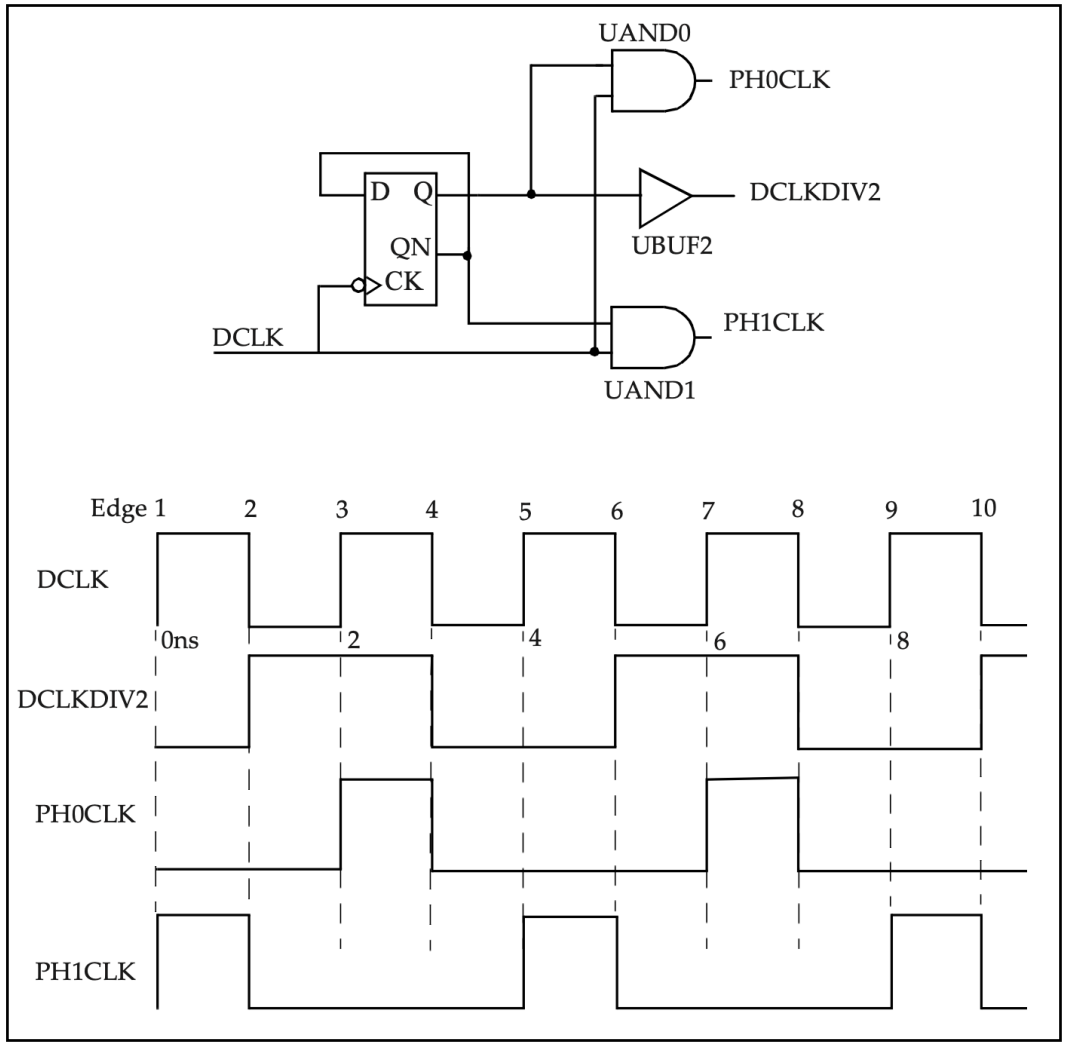
GENERATED CLOCKS

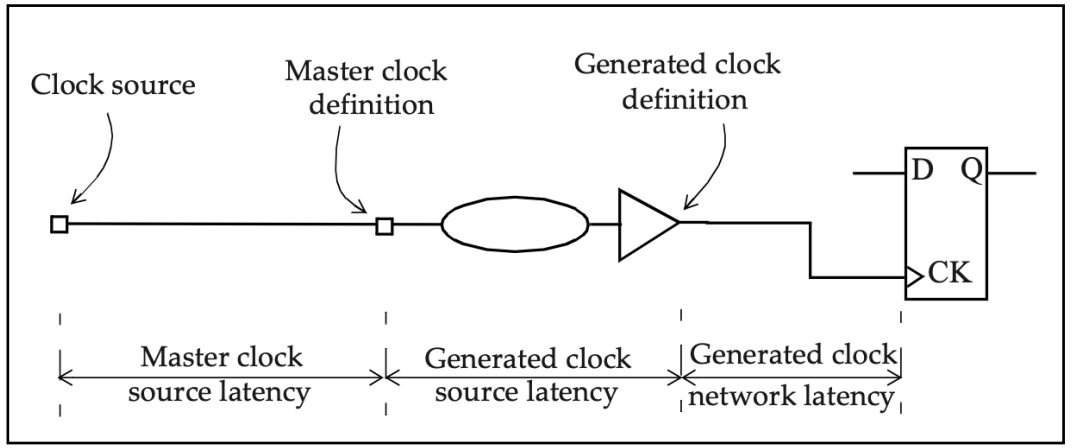
A generated clock is a clock derived from a master clock. A master clock is a clock defined using the create_clock specification. When a new clock is generated in a design that is based on a master clock, the new clock can be defined as a generated clock. For example, if there is a divide-by-3 circuitry for a clock, one would define a generated clock definition at the output of this circuitry. This definition is needed as STA does not know that the clock period has changed at the output of the divide-by logic, and more importantly what the new clock period is

```
create_clock -name CLKP 10 [get_pins UPLL0/CLKOUT]
  # Create a master clock with name CLKP of period 10ns
  # with 50% duty cycle at the CLKOUT pin of the PLL.
create_generated_clock -name CLKPDIV2 -source UPLL0/CLKOUT \
  -divide_by 2 [get_pins UFF0/Q]
  # Creates a generated clock with name CLKPDIV2 at the Q
  # pin of flip-flop UFF0. The master clock is at the CLKOUT
  # pin of PLL. And the period of the generated clock is double
  # that of the clock CLKP, that is, 20ns.
```



GENERATED CLOCKS contd.



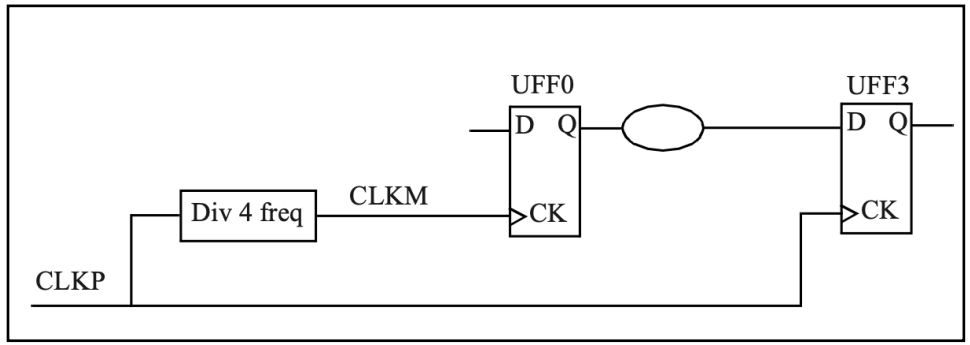


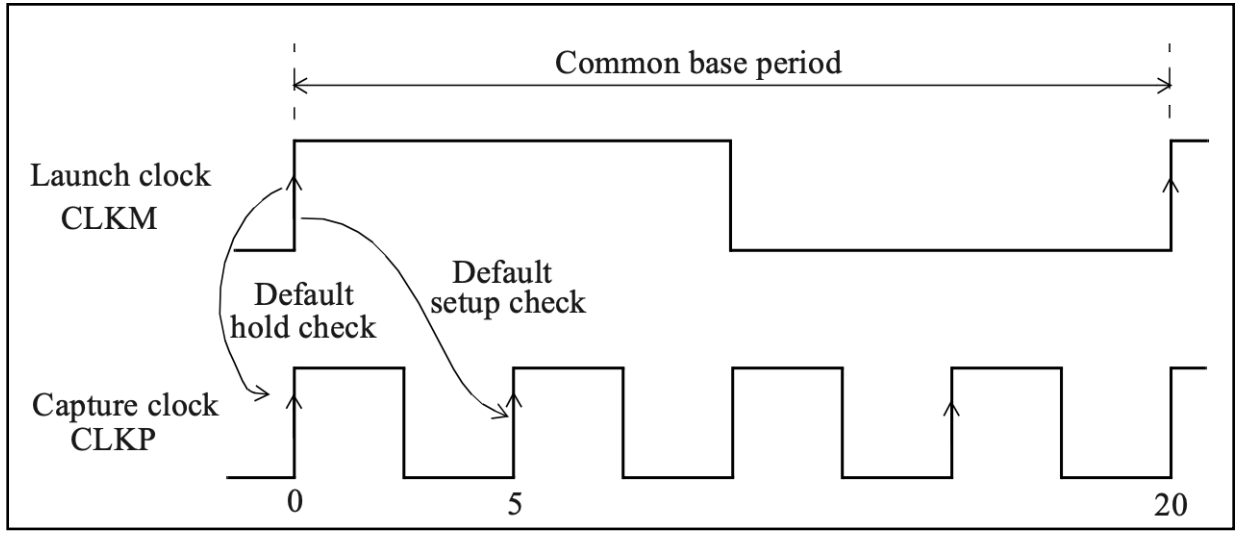
Latency on Generated Clock

Timing Across Clock Domains

SLOW TO FAST CLOCK DOMAINS — When the clock frequencies are different for the launch flip-flop and the capture flip-flop, STA is performed by first determining a common base period. An example of a message produced when STA is performed on such a design with the above two clocks is given below. **The faster clock is expanded so that a common period is obtained.**

```
create_clock -name CLKM \
  -period 20 -waveform {0 10} [get_ports CLKM]
create_clock -name CLKP \
  -period 5 -waveform {0 2.5} [get_ports CLKP]
```

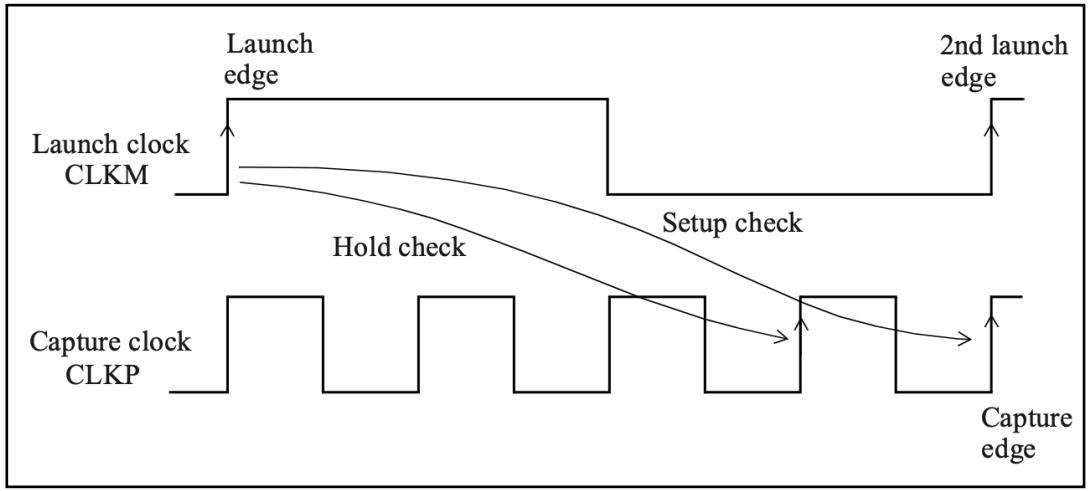




Timing Across Clock Domains + MCP

In the above example, we can see that the launch data is available every fourth cycle of the capture clock. Let us assume that the intention is not to capture data on the very next active edge of CLKP, but to capture on every 4th capture edge. This assumption gives the combinational logic between the flip-flops four periods of CLKP to propagate, which is 20ns. We can do this by setting the following multicycle specification:

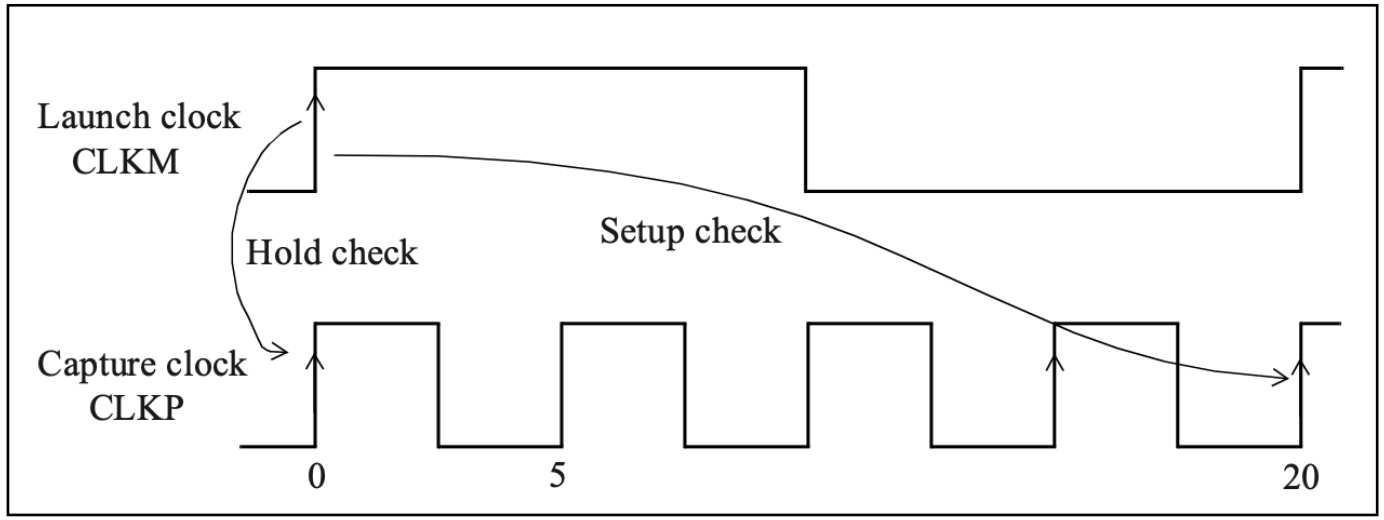
```
set_multicycle_path 4 -setup \
  -from [get_clocks CLKM] -to [get_clocks CLKP] -end
```



Problem with the case above

There is a problem with the case presented in the previous slide. The Setup Check Passes. However, Hold Check fails. Why? — It's because the hold check is derived from the setup check and defaults to one cycle preceding the intended capture edge. (Required Time is very high and Arrival Time is very low). We need to adjust the hold check edges.

```
set_multicycle_path 3 -hold \
  -from [get_clocks CLKM] -to [get_clocks CLKP] -end
```

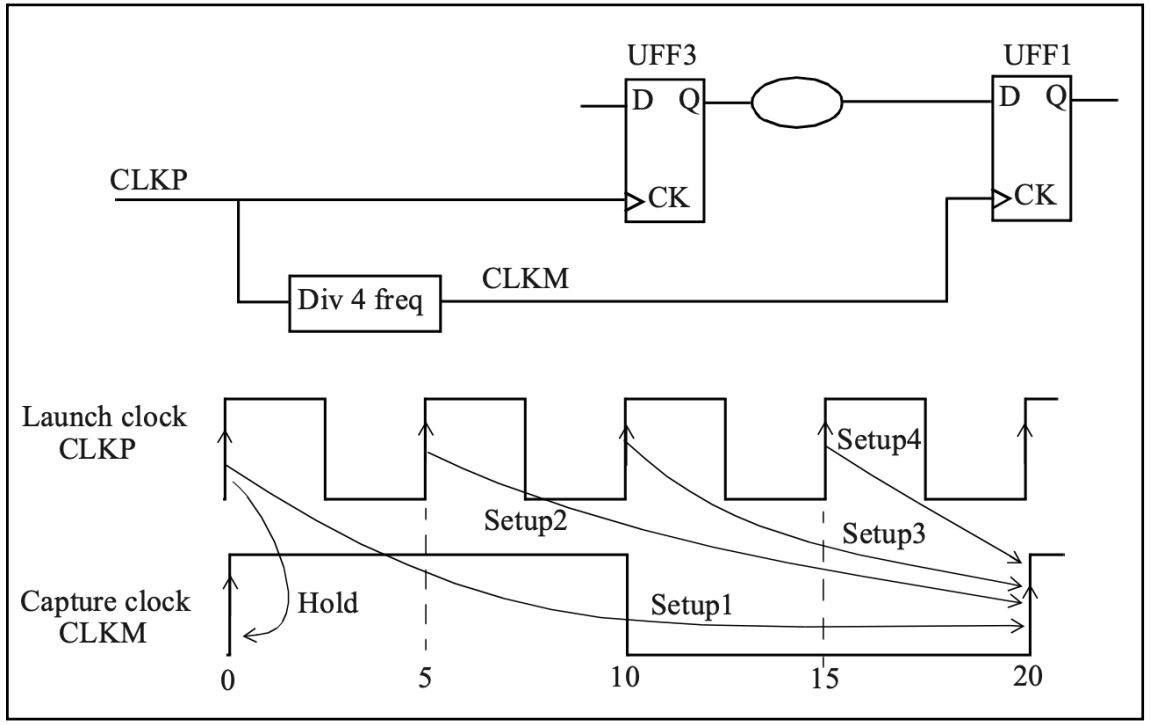


Adjusted Hold Check

Timing Across Clock Domains

FAST TO SLOW CLOCK DOMAINS — In this subsection, we consider examples where the data path goes from a fast clock domain to a slow clock domain. The most restrictive path is Setup(4) given in the figure

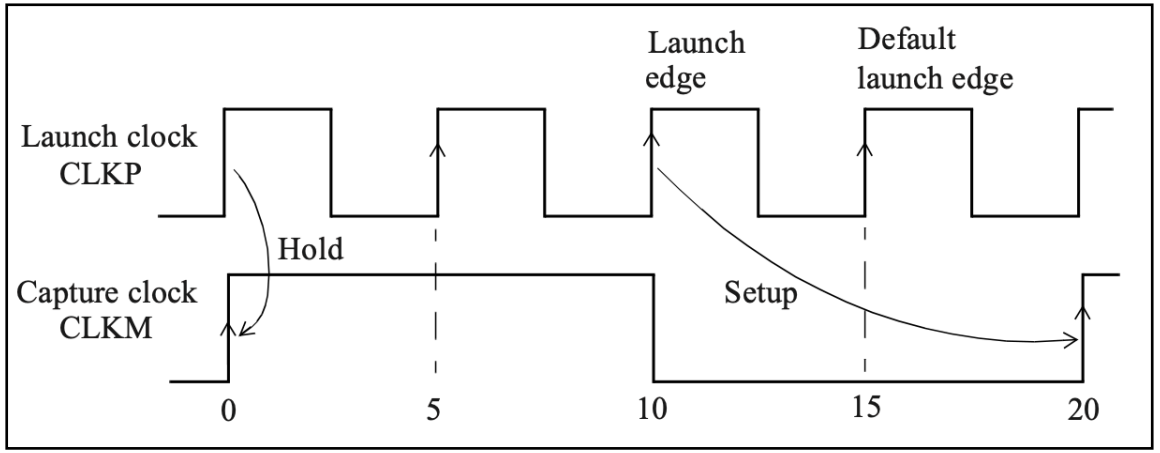
```
create_clock -name CLKM \
  -period 20 -waveform {0 10} [get_ports CLKM]
create_clock -name CLKP \
  -period 5 -waveform {0 2.5} [get_ports CLKP]
```



Timing Across Clock Domains + MCP

FAST TO SLOW CLOCK DOMAINS — In general, a designer may specify the data path from the fast clock to the slow clock to be a multicycle path. If the setup check is relaxed to provide two cycles of the faster clock for the data path, the following is included for this multicycle specification. The -start option specifies that the unit for the number of cycles (2 in this case) is that of the launch clock (CLKP in this case). The setup multi- cycle of 2 moves the launch edge one edge prior to the default launch edge, that is, at 10ns instead of the default 15ns. The hold multicycle ensures that the capture of the earlier data can reliably occur at 0ns due to the launch edge also at 0ns.

```
set_multicycle_path 2 -setup \
  -from [get_clocks CLKP] -to [get_clocks CLKM] -start

set_multicycle_path 1 -hold \
  -from [get_clocks CLKP] -to [get_clocks CLKM] -start
# The -start option refers to the launch clock and is
# the default for a multicycle hold.
```

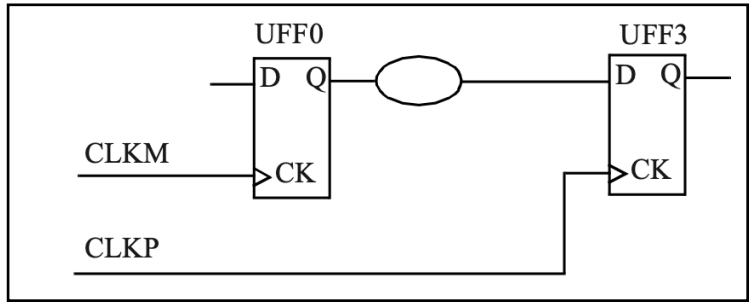


Setup multicycle of 2.

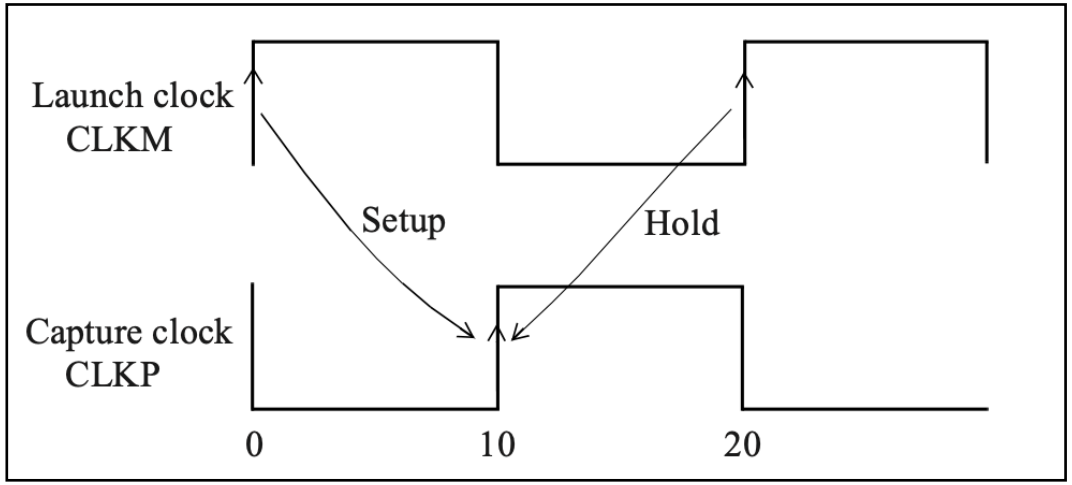
Timing Across Clock Domains (diff. Clock source)

SU & H Checks — Half Cycle Paths (Case 1)

```
create_clock -name CLKM \
  -period 20 -waveform {0 10} [get_ports CLKM]
create_clock -name CLKP \
  -period 20 -waveform {10 20} [get_ports CLKP]
```



- Always Remember — Edges For Hold Check
- **(L+1 —> C or L —> C-1) (L- launch, C- capture)**

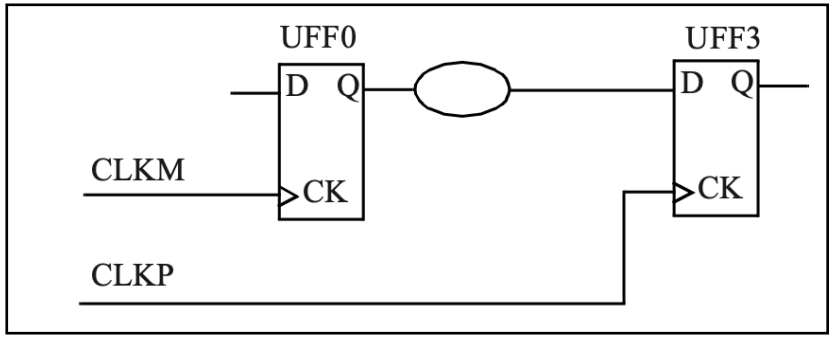


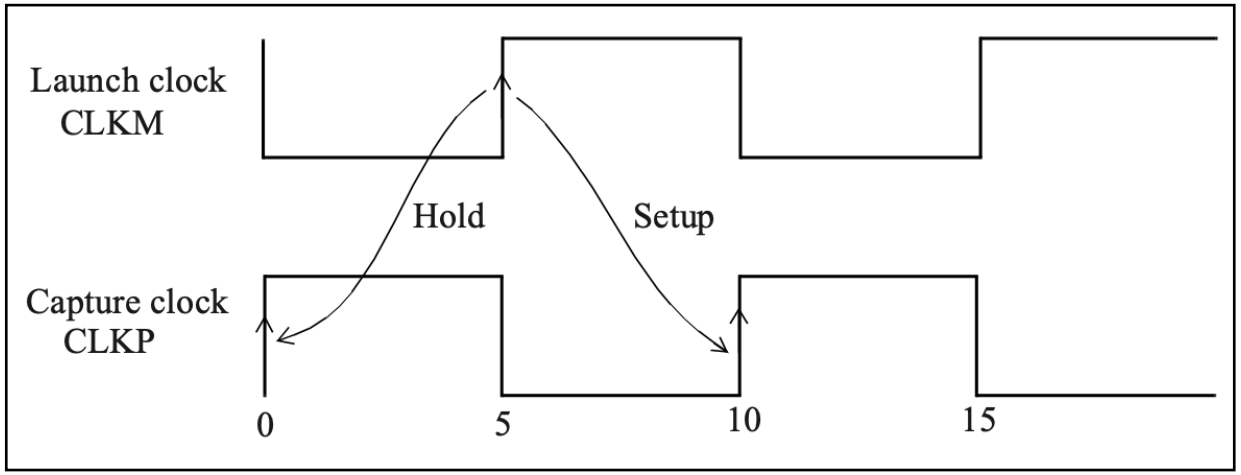
Timing Across Clock Domains (diff. Clock source)

SU & H Checks — Half Cycle Paths (Case 2)

```
create_clock -name CLKM \
  -period 10 -waveform {5 10} [get_ports CLKM]
```

```
create_clock -name CLKP \
  -period 10 -waveform {0 5} [get_ports CLKP]
```



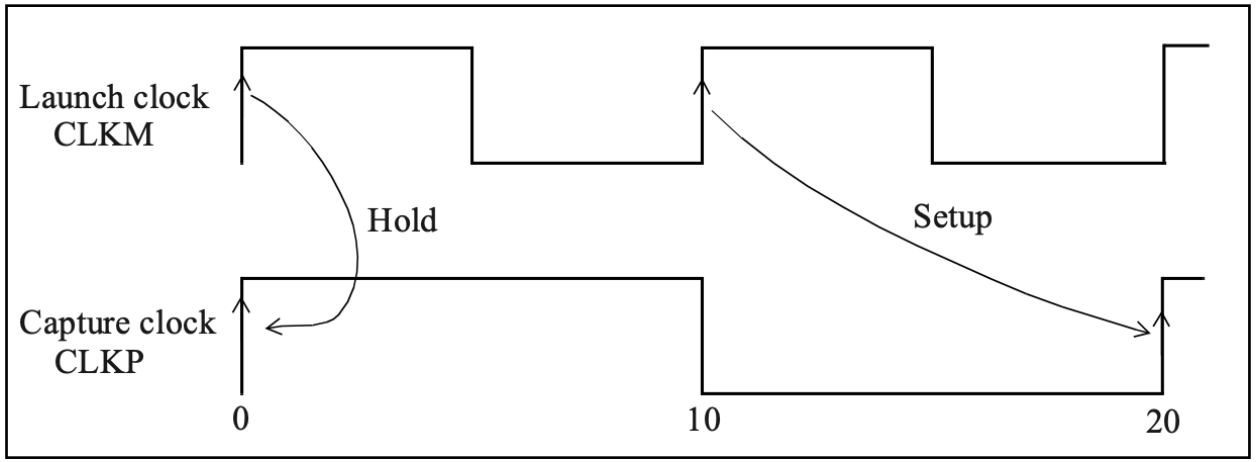


Timing Across Clock Domains (diff. Clock source)

FAST TO SLOW CLOCK DOMAINS — SU & Hold Checks

The setup check is from the launch edge at 10ns to the capture edge at 20ns. The hold check is from the launch edge at 0ns to the capture edge at 0ns

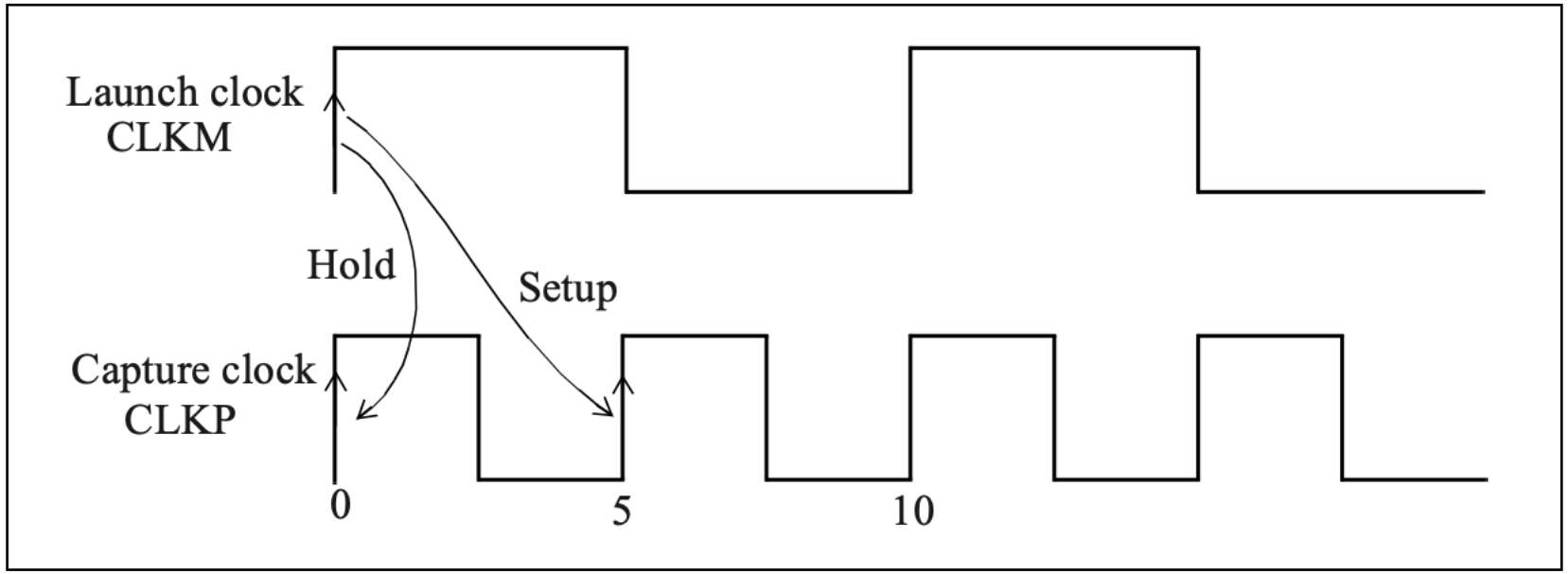
```
create_clock -name CLKM \
  -period 10 -waveform {0 5} [get_ports CLKM]
create_clock -name CLKP \
  -period 20 -waveform {0 10} [get_ports CLKP]
```



Timing Across Clock Domains (diff. Clock source)

SLOW TO FAST CLOCK DOMAINS — SU & Hold Checks

The setup check is from the launch edge at 0ns to the capture edge at 5ns. The hold check is from the launch edge at 0ns to the capture edge at 0ns

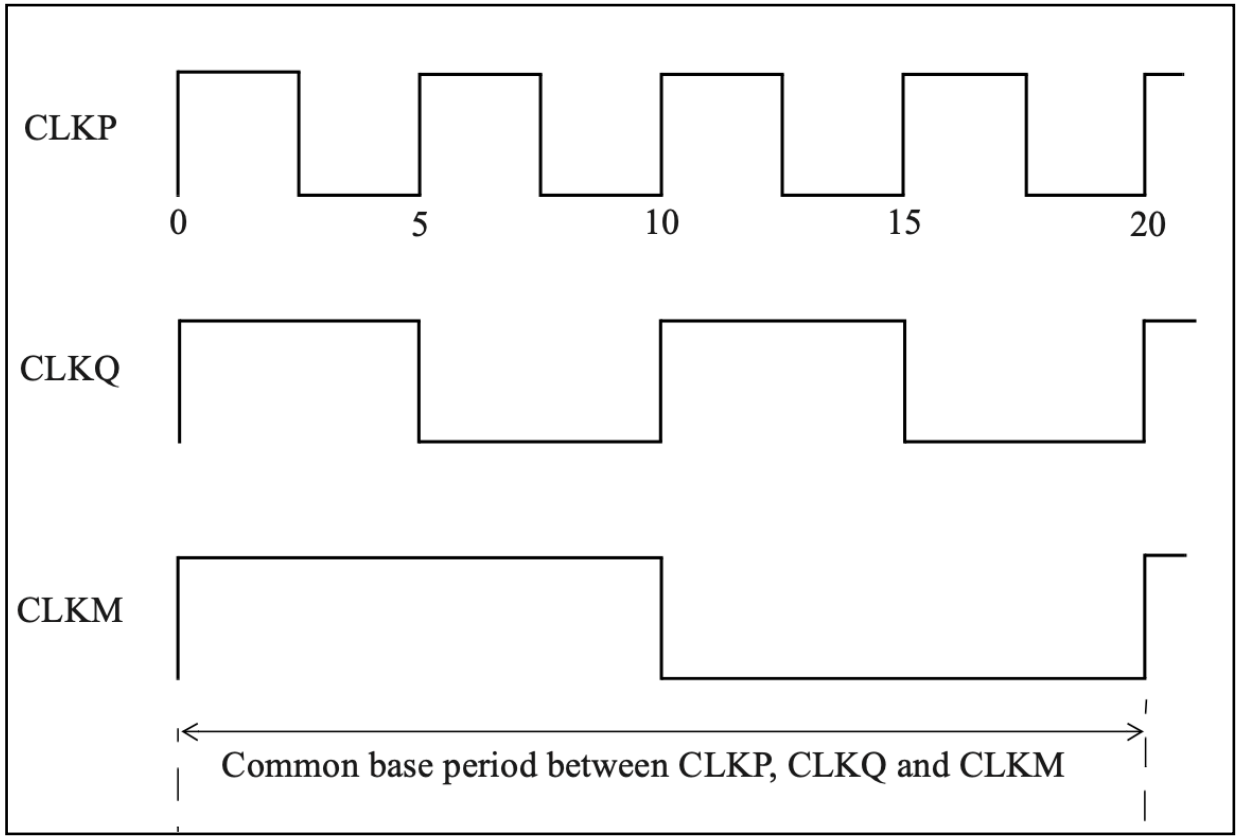


MULTIPLE CLOCKS

INTEGER MULTIPLES

Often there are multiple clocks defined in a design with frequencies that are simple (or integer) multiples of each other. In such cases, STA is per- formed by computing a common base period among all related clocks (two clocks are related if they have a data path between their domains). The common base period is established so that all clocks are synchronized.

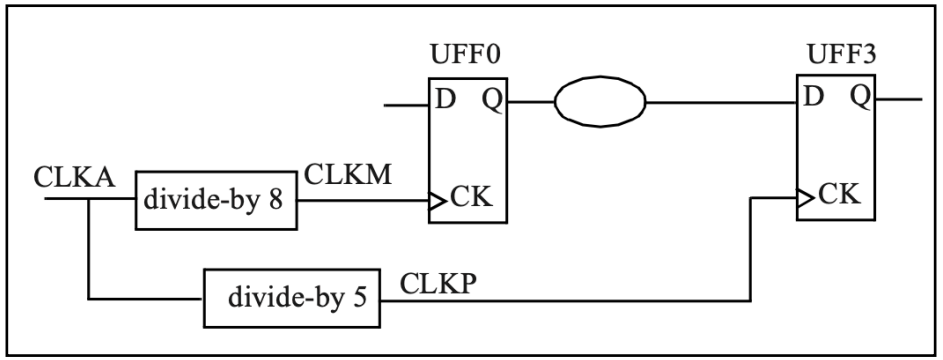
```
create_clock -name CLKM \
  -period 20 -waveform {0 10} [get_ports CLKM]
create_clock -name CLKQ -period 10 -waveform {0 5}
create_clock -name CLKP \
  -period 5 -waveform {0 2.5} [get_ports CLKP]
```

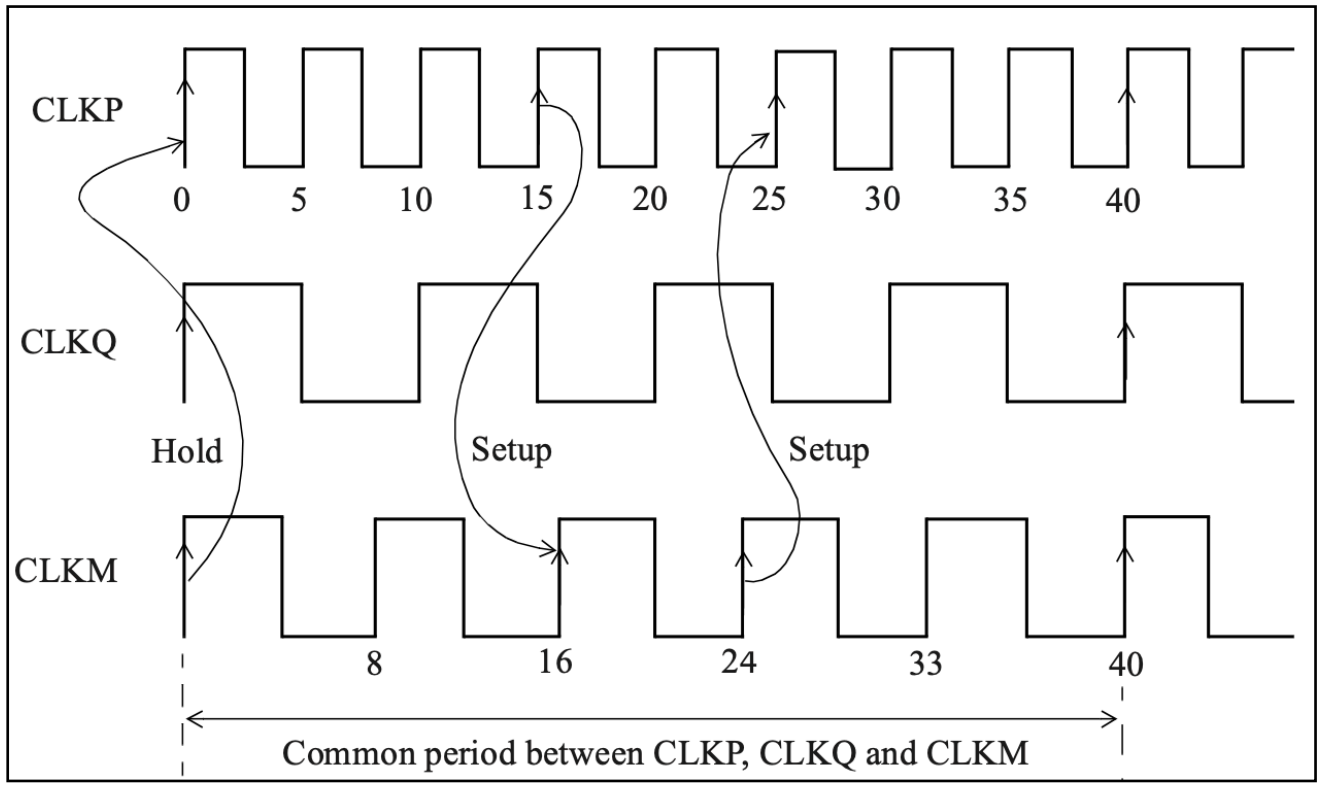


MULTIPLE CLOCKS

NON-INTEGER MULTIPLES

Consider the case when there is a data path between two clock domains whose frequencies are not multiples of each other. For example, the launch clock is divide-by-8 of a common clock and the capture clock is divide-by-5 of the common clock



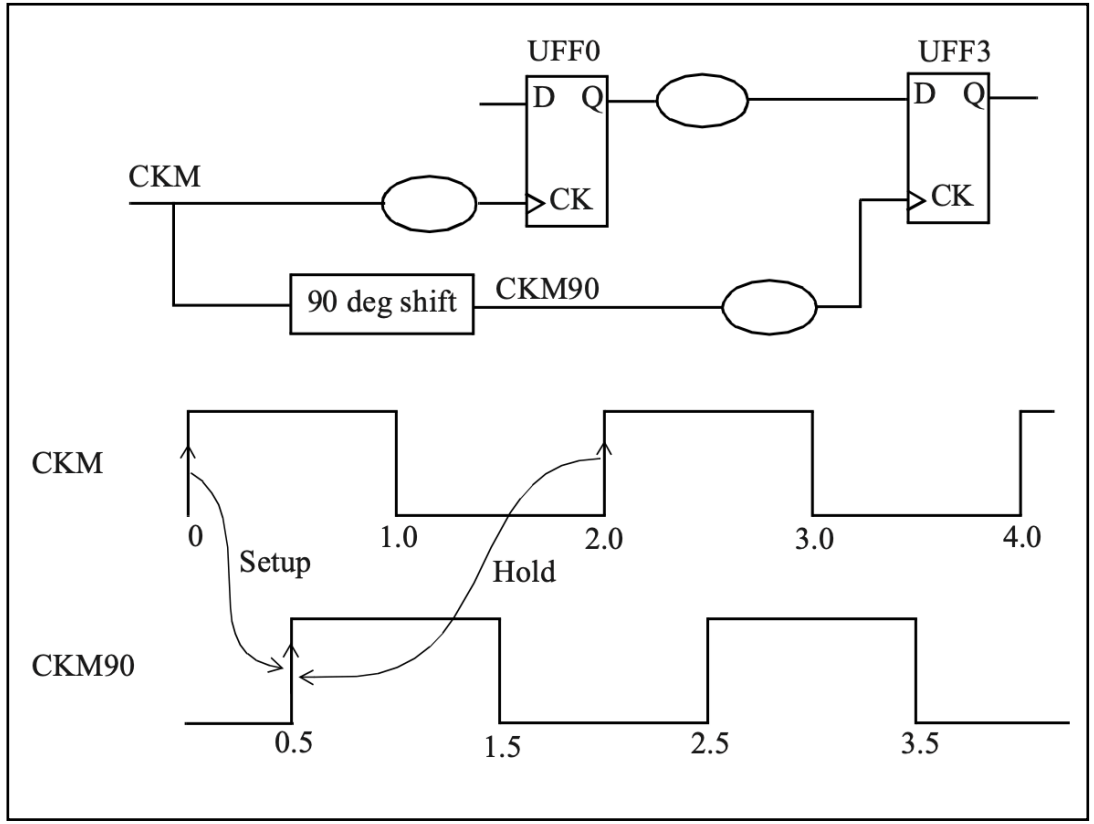


MULTIPLE CLOCKS

PHASE SHIFTED CLOCKS

Here is an example where two clocks are ninety degrees phase-shifted with respect to each other. There are still a lot of things left to cover. (Derating, Crosstalk Analysis). **Latch Timing is much harder than what was covered so far.**

```
create_clock -period 2.0 -waveform {0 1.0} [get_ports CKM]
create_clock -period 2.0 -waveform {0.5 1.5} \
  [get_ports CKM90]
```



CLOCK UNCERTAINTY

— Clock uncertainty is the difference between the arrivals of clocks at registers in one clock domain or between domains. it can be classified as static and dynamic clock uncertainties.

— Timing Uncertainty of clock period is set by the command **set_clock_uncertainty** at the synthesis stage to reserve some part of the clock period for uncertain factors (like skew, jitter, OCV, CROSS TALK, MARGIN or any other pessimism) which will occur in PNR stage. The uncertainty can be used to model various factors that can reduce the clock period.

It can be defined for both setup and hold.

```
set_clock_uncertainty –setup 0.2 [get_clocks CLK_CONFIG]
set_clock_uncertainty –hold 0.05 [get_clocks CLK_CONFIG]
```